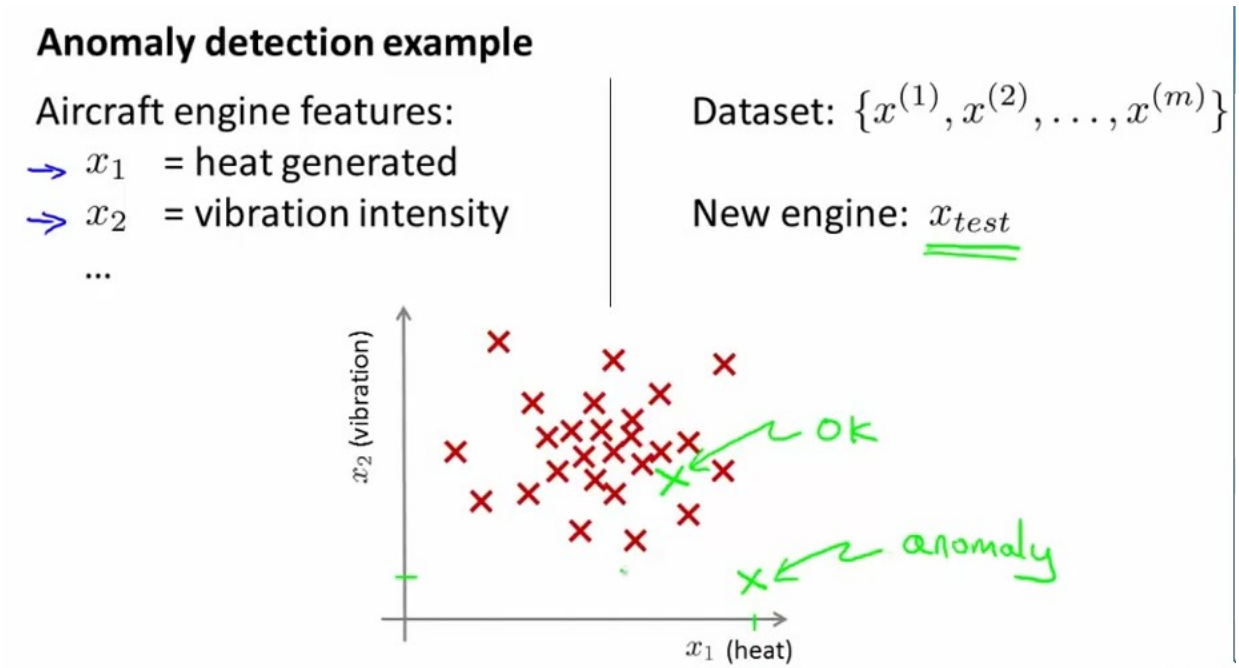
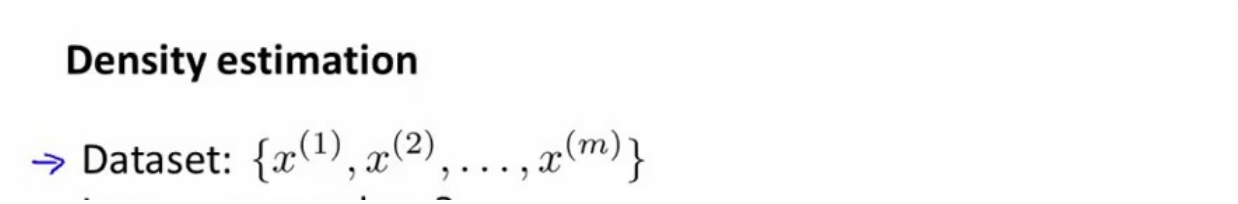
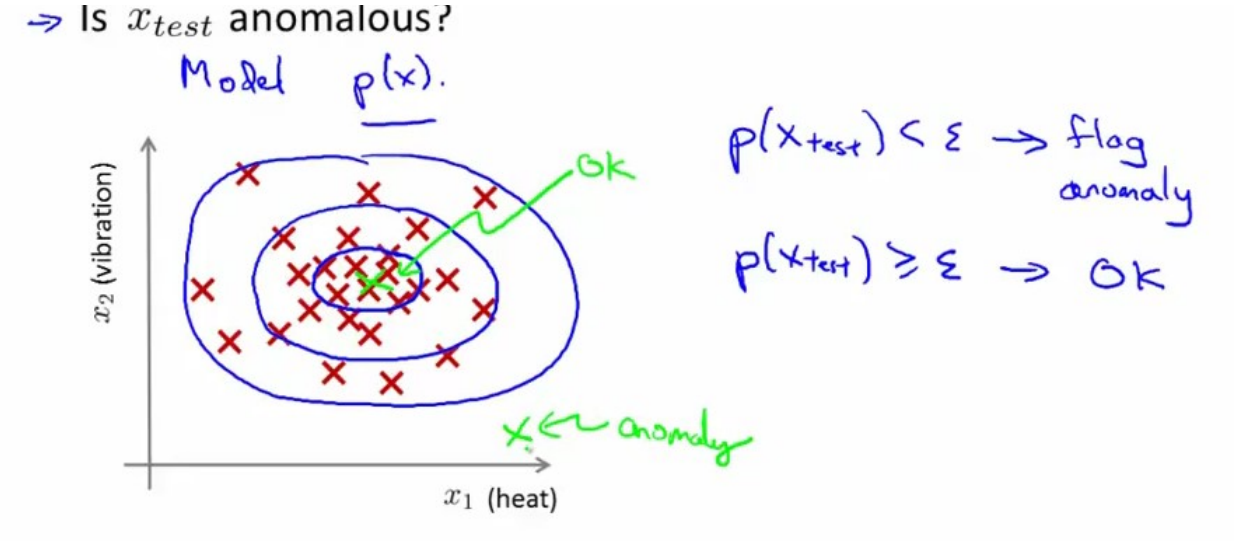
ML 09: ANOMALY DETECTION, RECOMMENDER SYSTEM

이번시간엔 *anomaly detection* 과 *recommender system* 을 배운다.

Anomaly Dectection





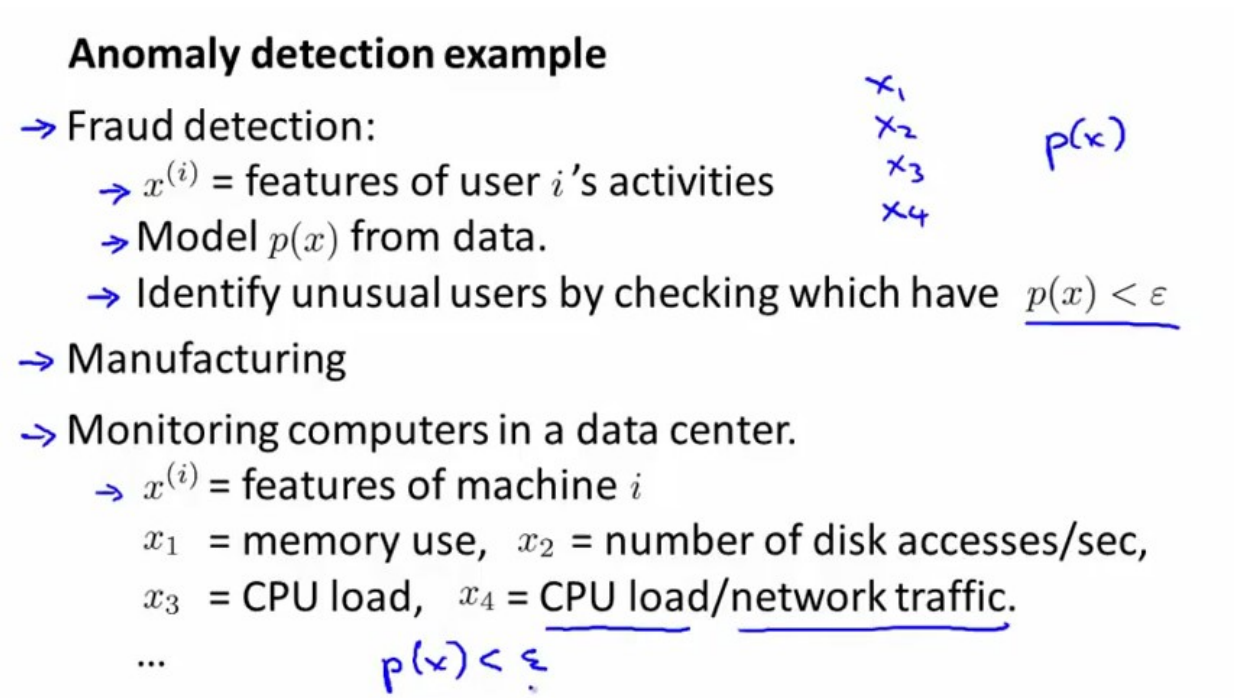


(http://blog.csdn.net/linuxcumt1)

anomaly 는 정상집단에서 떨어진 데이터라 보면 된다. 공장에서 품질이 떨어지는 제품을 골라낼때 사용할 수 있는데, 위 그림은 비행기 엔진 공장을 예로 들어 설명한다.

데이터로부터 `p(x)` 를 만들어, 검사할 데이터가 *threshold* 를 넘는지 안넘는지 검사해 *anomaly* 로 판정할 수 있다.

참고로, *anomaly* 가 너무 많으면, *false positive* 가 높은 것인데 이 때는 *threshold* 를 줄이면 된다.

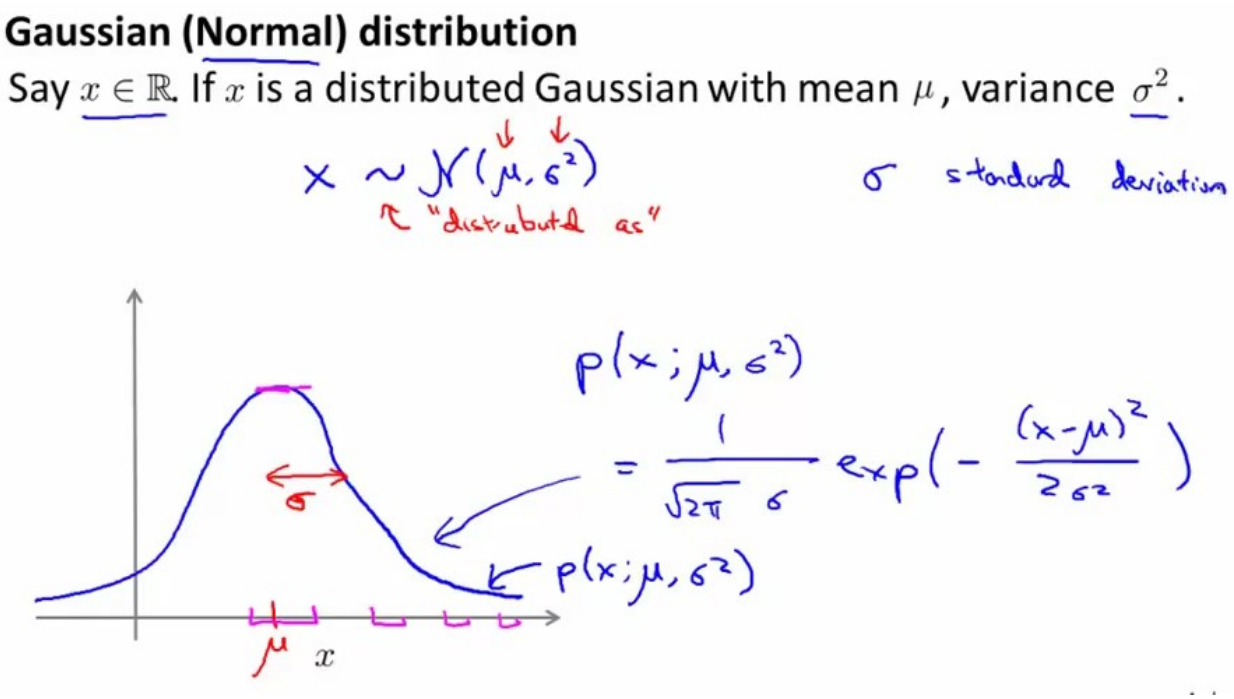


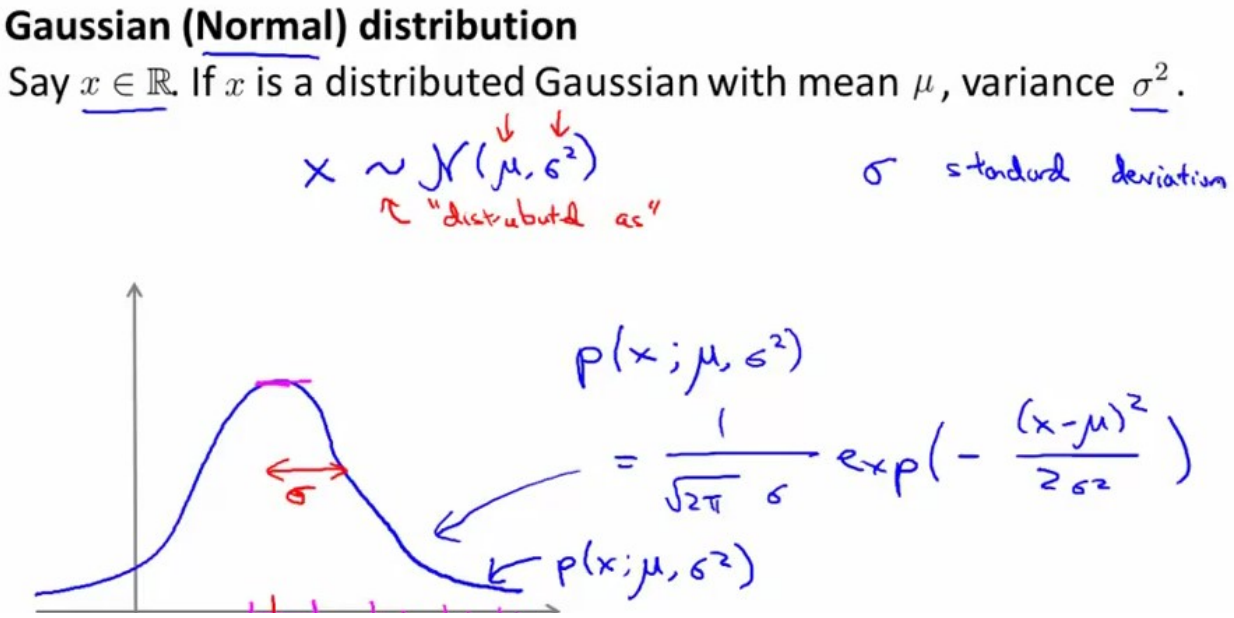
(http://blog.csdn.net/linuxcumt1)

anomaly detection 은 *fraud detection* 에 많이 사용된다. 데이터로부터 모델 `p(x)` 를 만들고 *unusual user* 를 검사하기 위해 `p(x) < e` 인지 검사하면 된다.

이외에도 항공기 엔진 예제처럼 제품의 품질 관리나, 데이터 센터에서의 노드 과부하 탐지등에 사용할 수 있다.

Gaussian Distribution

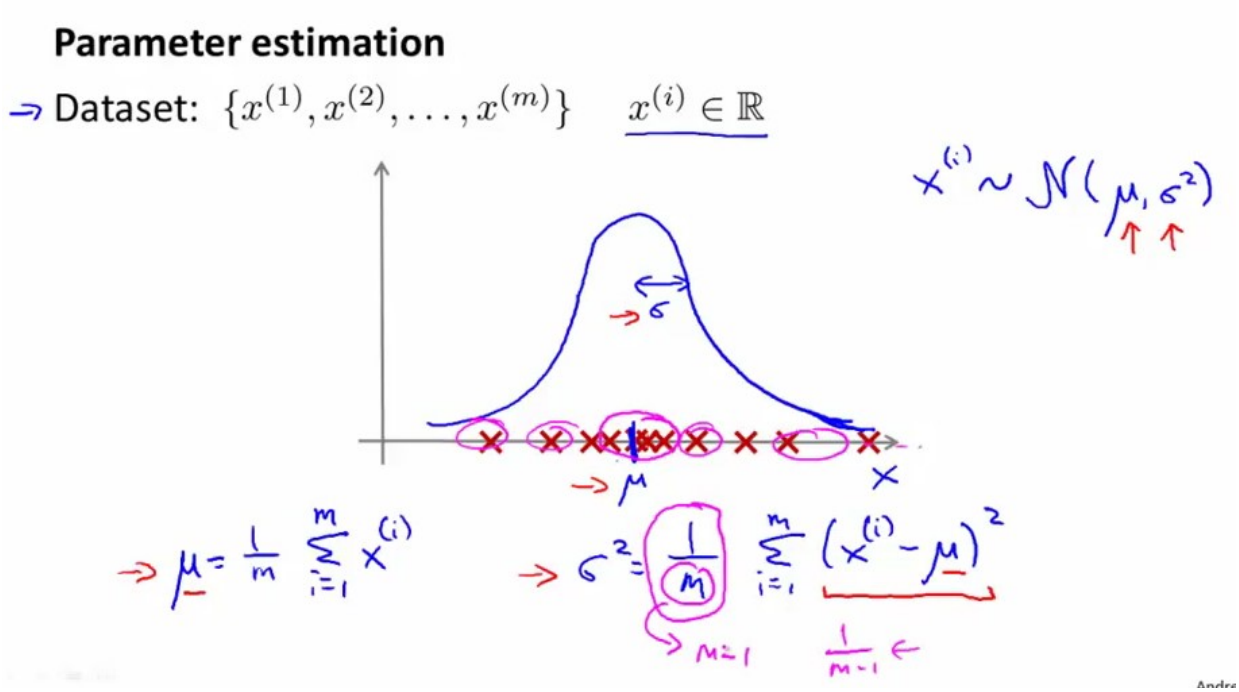




(http://blog.csdn.net/linuxcumt1)

gaussian density 공식은

$$P(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\ \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$



(http://blog.csdn.net/linuxcumt1)

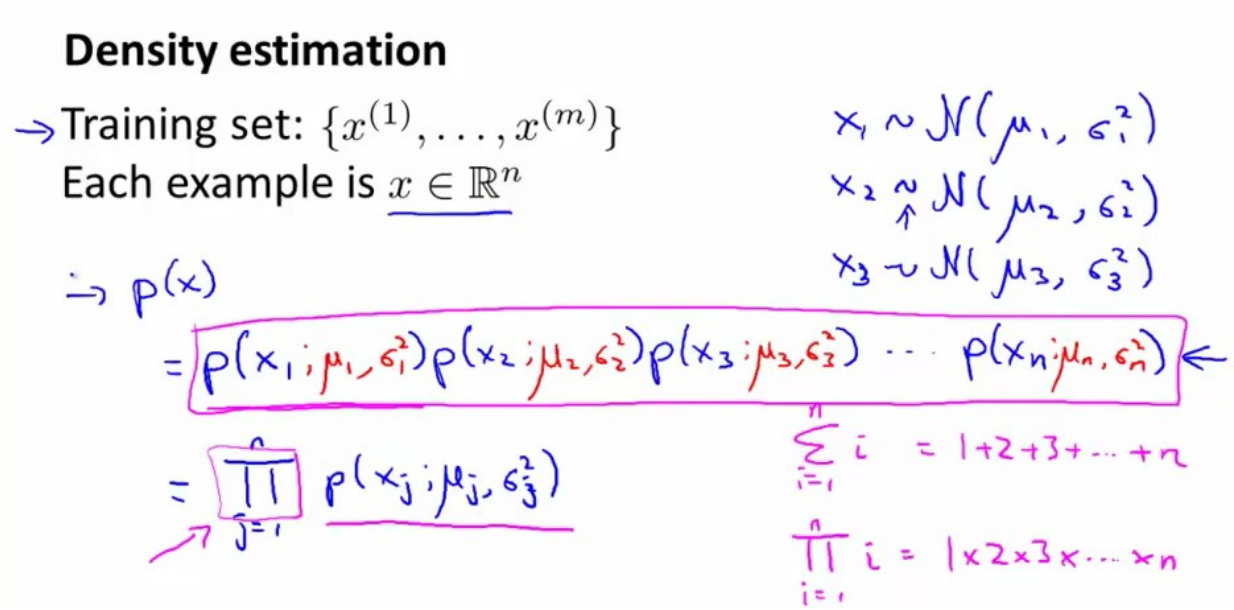
평균과 분산은

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)$$

$$\cdots \quad i=1$$

Anomaly Detection Algorithm



(http://blog.csdn.net/linuxcumt1)

각 *feature* 가 가우시안 분포를 따른다고 하면,

$$\begin{aligned} & p(x) \\ &= p(x_1; \mu_1, \sigma_1^2)\ p(x_2; \mu_2, \sigma_1^2) \cdots\ p(x_n; \mu_n, \sigma_1^n) \\ &= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) \end{aligned}$$

이렇게 가정하려면, 각 *feature* 가 독립적이어야 하지만 실제로는 독립적이지 않더라도 어느정도 동작한다. 이 때

$$\mu_i = \frac{1}{m} \sum^{m} x^{(i)}$$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

Anomaly detection algorithm

1. Choose features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

3. Given new example x, compute $p(x)$:

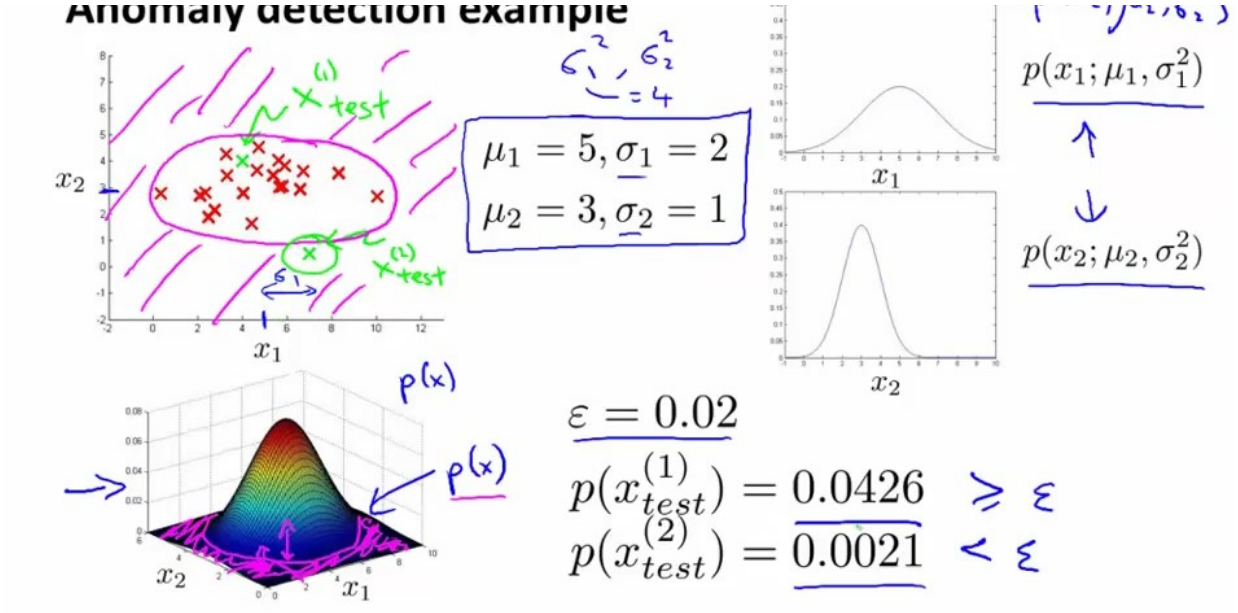
$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

(http://blog.csdn.net/linuxcumt1)

따라서 `p(x)` 는 아래 식이 된다. `p(x)` 는 *feature* 가 나올 확률로 이해하면 된다. 이 때 `p(x)` 가 상당히 작으면, 평균에 가깝지 않은 *feature* 가 많이 나왔다는 뜻이므로 *anomaly* 라 볼 수 있다.

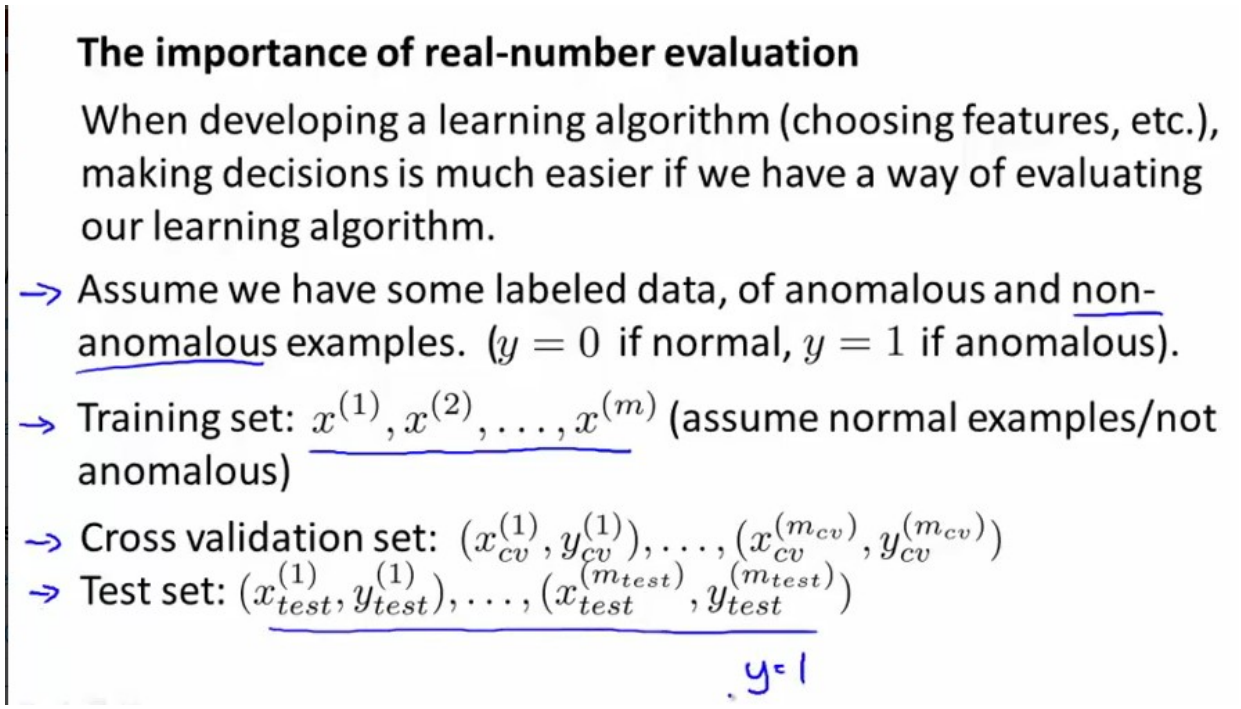
$$p(x) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

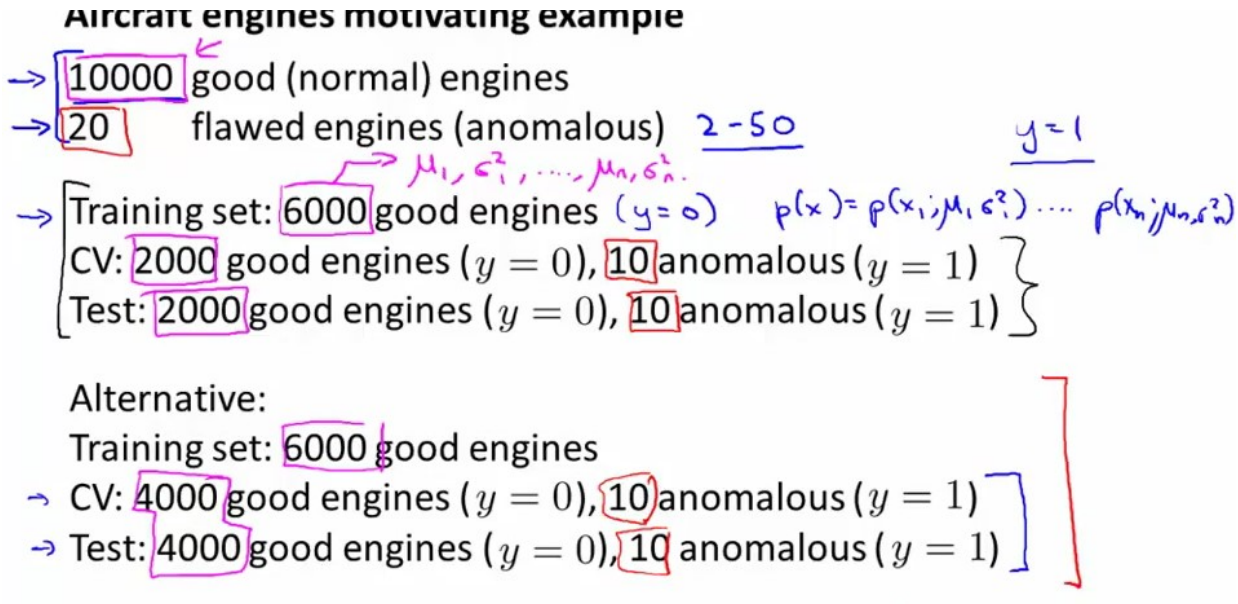


(http://blog.csdn.net/linuxcumt1)

두 *feature* `x1, x2` 의 가우시안 분포를 3차원으로 조합하면 `p(x)` 가 좌측 하단 3차원 원뿔의 높이가 된다.

Evaluating Anomaly Detection



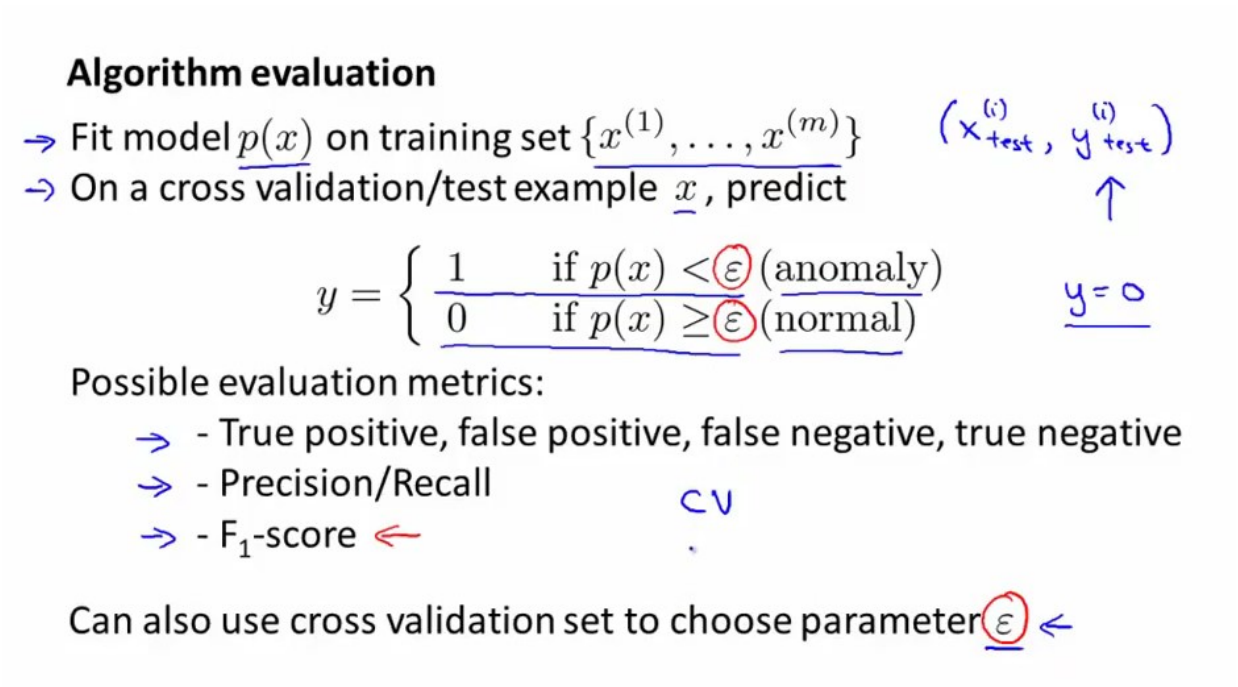


(http://blog.csdn.net/linuxcumt1)

anomaly 를 잘 나타낼거 같은 *feature* 를 골라내고, 이를 이용해 모델을 만든다.

우리가 가진 데이터가 *anomaly* 를 알려주는 y 가 있다면, 위 그림처럼 *training set* 으로 *non-anomalous* 을 이용하고, *CV, Test Set* 으로 나머지를 반반씩 분할하면 된다.

즉 *good example* 로 모델을 만들고, *anomaly* 가 섞여있는 *cv, test set* 으로 평가한다.



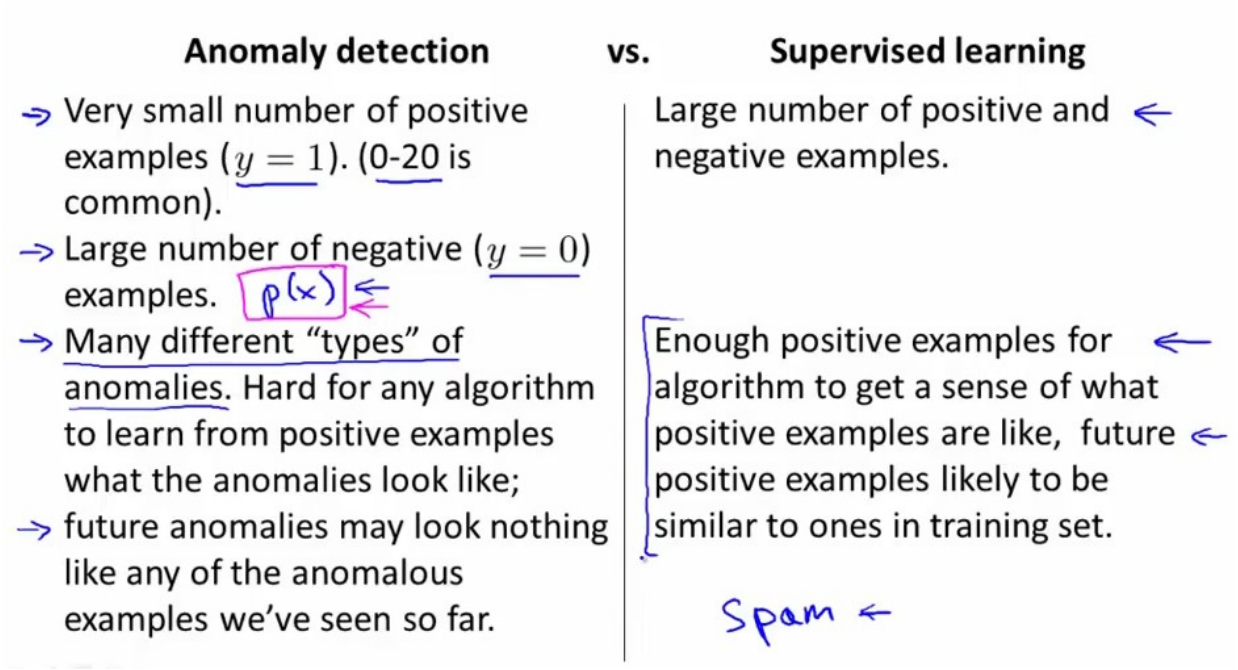
(http://blog.csdn.net/linuxcumt1)

이 때 *skewed classess* 이기 때문에 (`y = 0` 이 대다수, `y = 1` 은 희박) 단순히 정확도로 평가하긴 좀 무리가 있다. *precision, recall, f1 score* 등을 이용해 평가해야 한다.

threshold 인 `e` (엡실론) 를 고르기 위해 *cross validation* 을 이용할 수 있다. *f1 score* 를 최대화 하는 `e` 를 고른다거나.

Anomaly Dectection vs Supervised Learning

`y` 값이 있는 데이터라면, 왜 *supervised learning* 을 이용하지 않을까?



(http://blog.csdn.net/linuxcumt1)

Anomaly Detection

anomaly detection skewed class 가 있을 때 사용한다.

> Many different **types** of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like

> Future anomalies may look nothing like any of the anomalous examples we've seen so far

보면 알겠지만 *anomaly* 가 굉장히 다양할 수 있기 때문에 *anomaly* 를 특정 형태로 구분짓는 알고리즘을 쓰긴 좀 힘들다.

게다가, 가지고 있는 데이터 셋에서 보지 못했던 새로운 종류의 *anomaly* 가 나올 수도 있다.

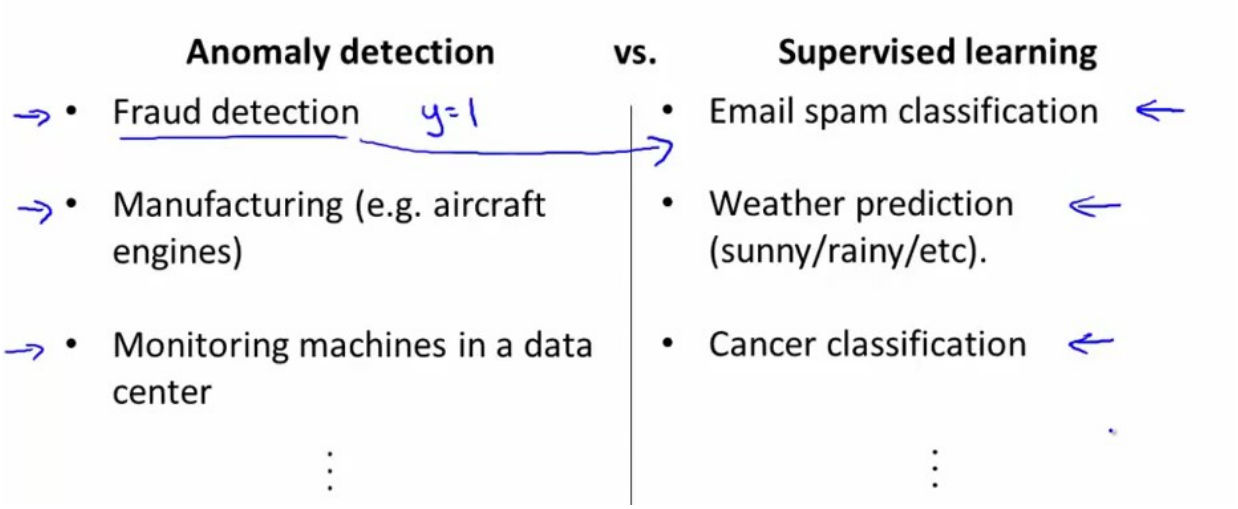
Supervised Learning

positive, negative example 이 많을 때 사용한다.

> Enough positive examples for algorithms to get a sense of what positive examples are like, futre positive example likly to be similar to ones in training set

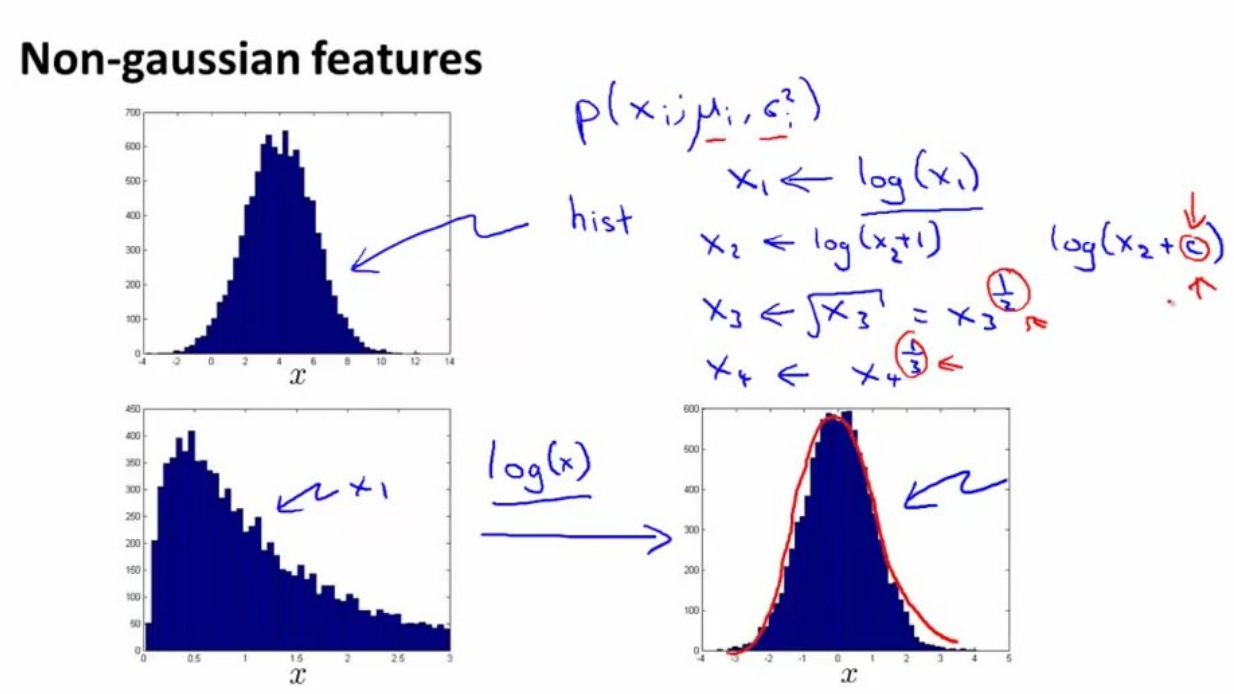
supervised learning 에서 *positive example* 은 어떤 특정 형태기 때문에, 미래에 발견할 *positive example* 도 비슷한 형태라 생각될 때 사용한다.

SPAM filtering 에서는 다양한 타입의 *positive example* 이 있어도, 우리가 충분한 양의 *positive example* 이 있기 때문에 커버할 수 있어 *supervised learning* 을 사용한다.



(http://blog.csdn.net/linuxcumt1)

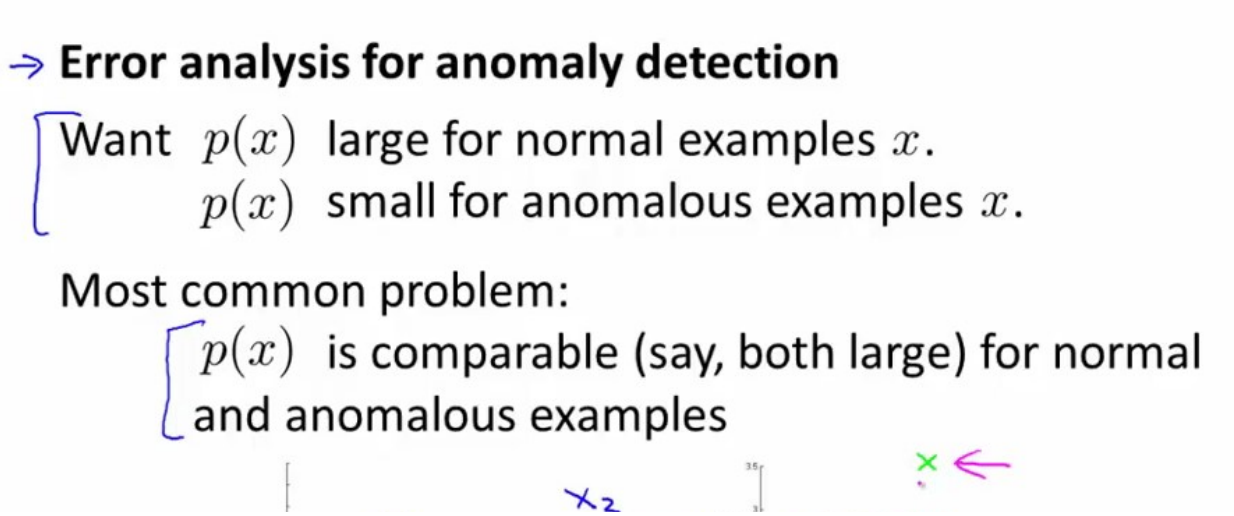
Choosing What Features to Use

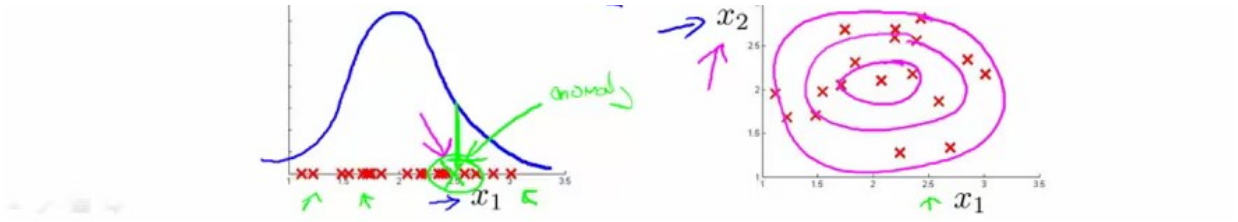


(http://blog.csdn.net/linuxcumt1)

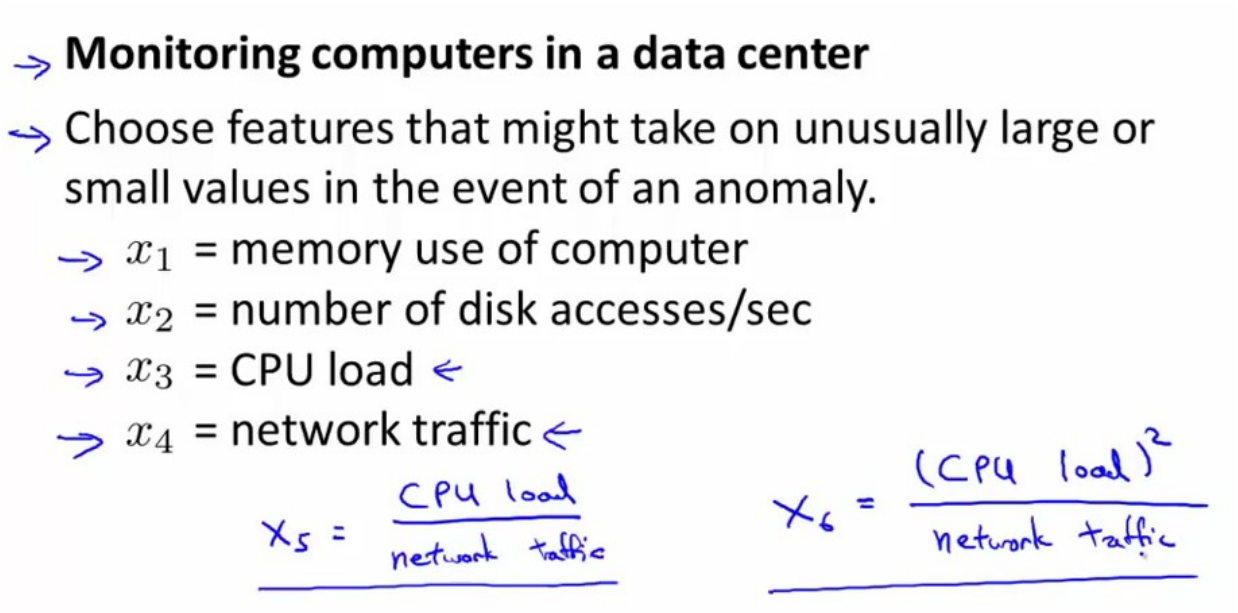
feature 의 분포가 가우시안이면 고맙지만, 아닐경우 변환이 필요하다. 왼쪽 아래 분포에 로그를 씌우면, 가우시안 분포 비슷하게 보인다.

다른 방법으로는 `log(x_2 + c)` , `sqrt(x_3)` 등등이 있다.





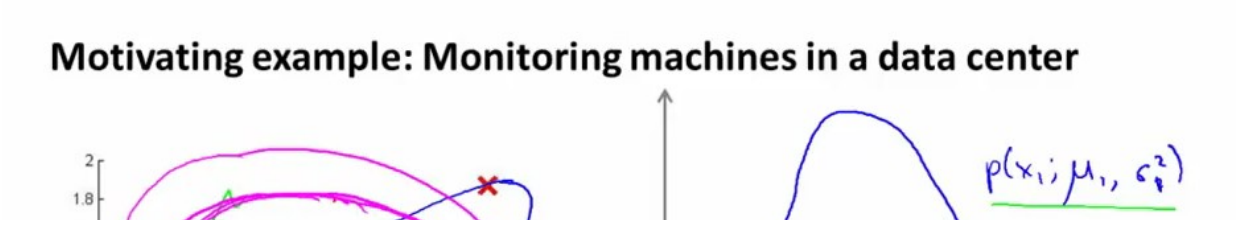
흔한 에러는 `p(x)` 가 *normal, anomalous* 에 대해서 모두 높은 경우인데, 슬라이드의 아래쪽에서 볼 수 있듯이 `x2` 라는 *feature* 를 만들어서 *anomaly* 를 발견하는 알고리즘을 만들 수 있다.

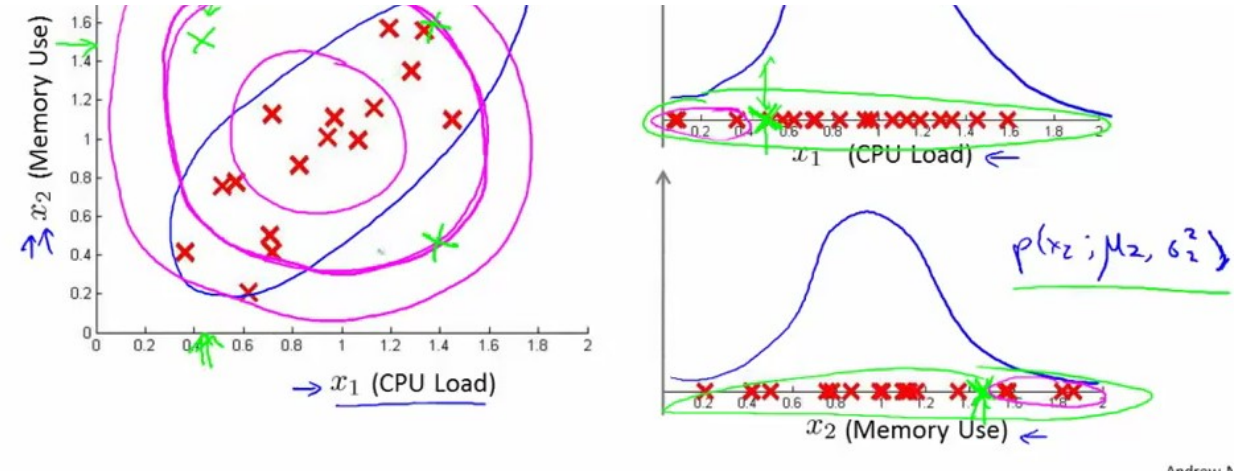


(http://blog.csdn.net/linuxcumt1)

anomaly 를 위한 *feature* 를 고를 때 특이하게 높거나, 낮을 수 있는 것을 고르면 된다. 데이터 센터 예제에서는 *CPU load / network traffic* 등이 있을 수 있다. 네트워크 트래픽이 낮은데 *CPU load* 가 높다면 확실히 *anomaly* 기 때문이다.

Multivariate Gaussian Distribution



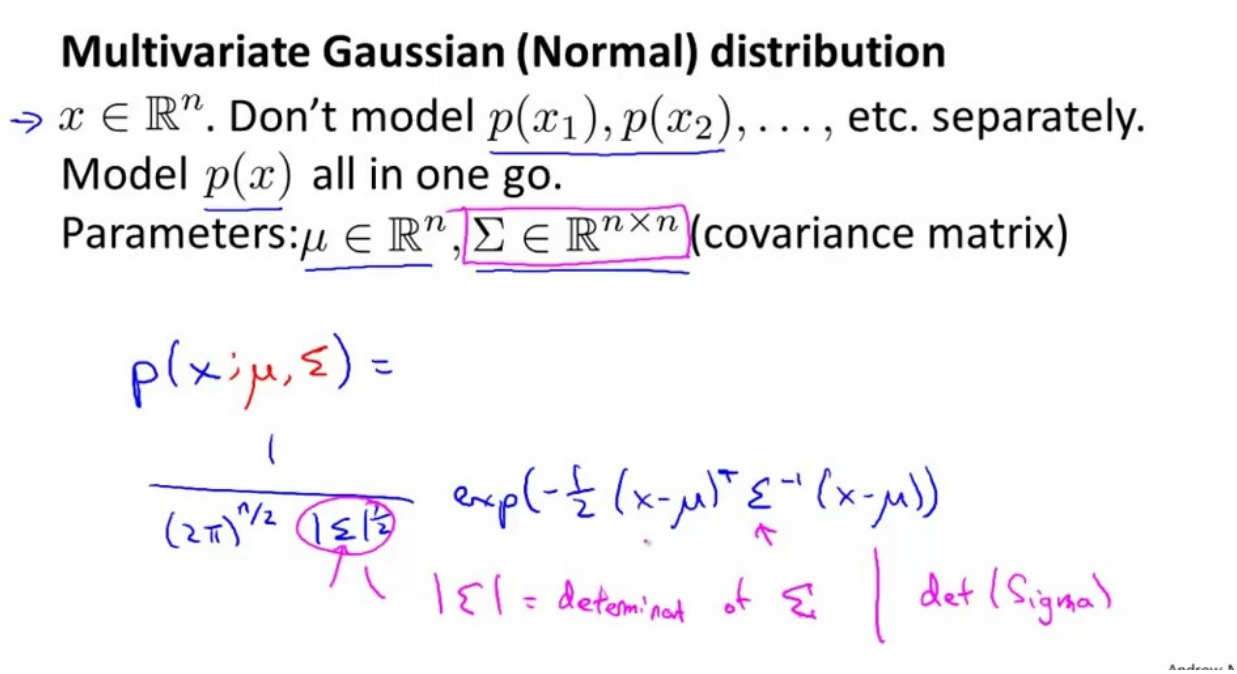


(http://blog.csdn.net/linuxcumt1)

feature 를 *CPU laod, memory use* 로 했을 때 낮은 CPU 부하에도 메모리 사용량이 높으면 *anomaly* 라 볼 수 있다.

그런데, 슬라이드의 왼쪽 그림에서 녹색으로 표시한 *anomaly* 는 지금까지 설명했던 알고리즘으로 찾기가 힘들다. 적당한 수준의 *memory use* 와 그리 낮지 않은 *cpu load* 를 가지기 때문이다.

실제 *normal example* 이 타원형이기 때문에, 원으로 *anomaly* 를 찾기는 어렵다.



따라서 `p(x_1)p(x_2)...` 을 이용한 모델 말고 다른 방법으로 모델을 만들어야 한다.

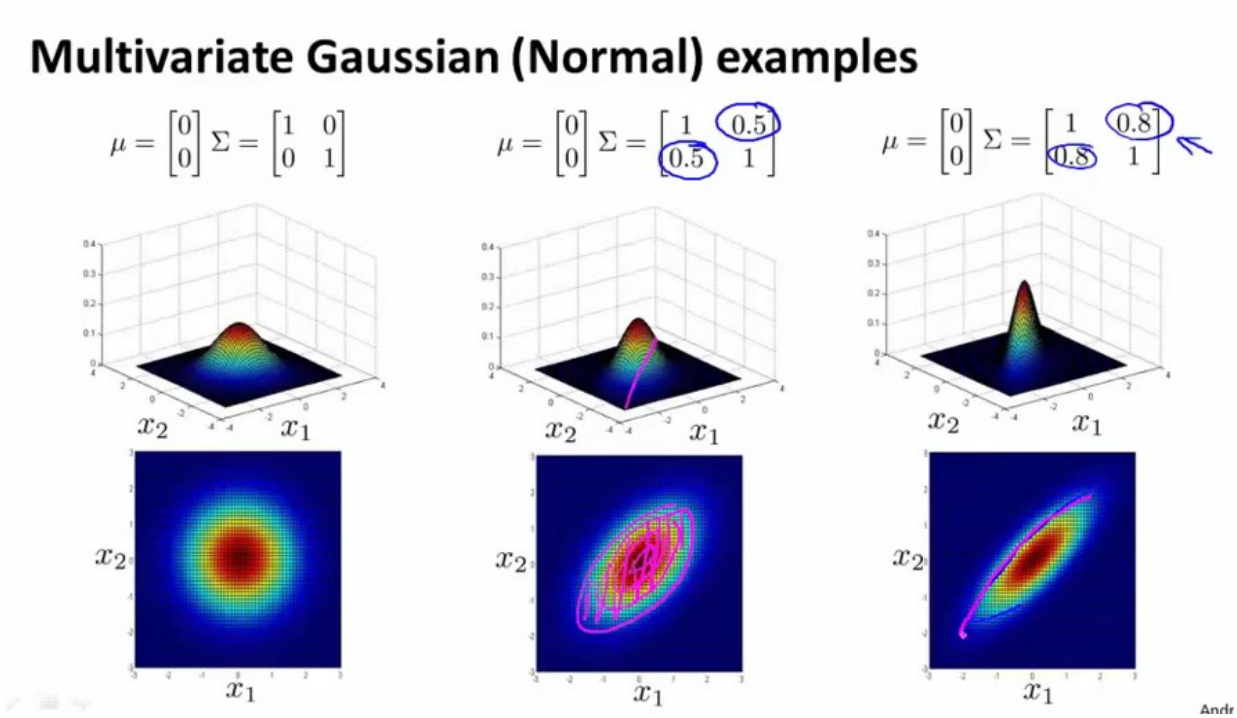
`u` 를 `n` 벡터라 하고, `Sigma` 를 `u` 의 *convariance matrix* 라 하자. 그러면

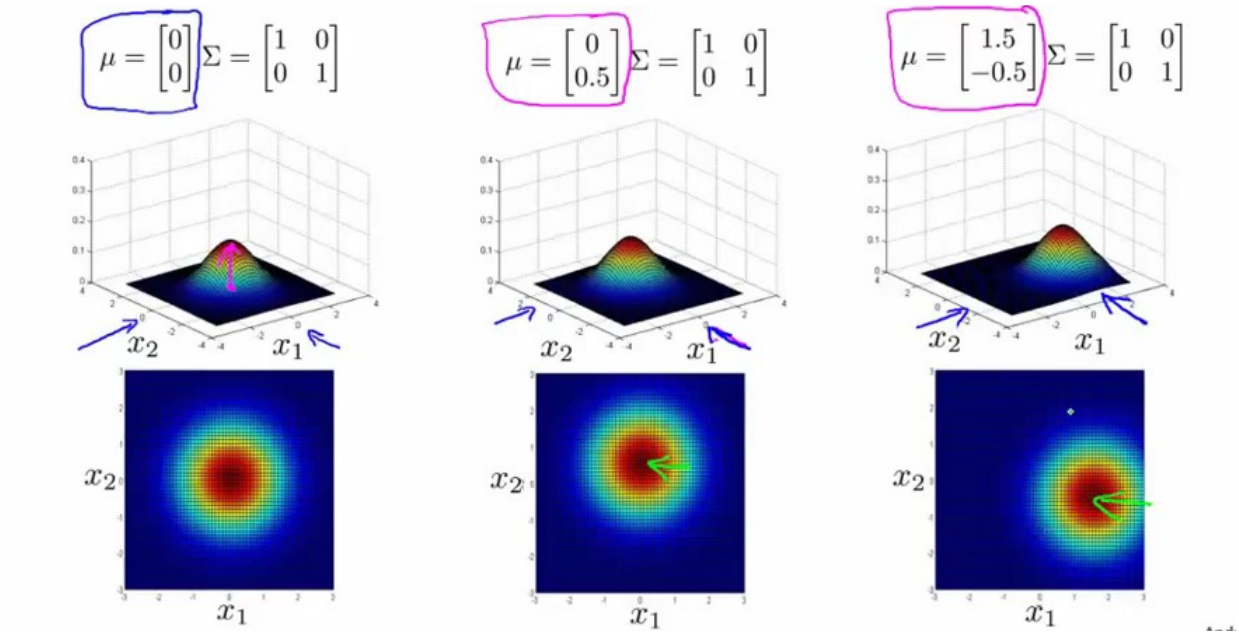
$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}\,|\Sigma|^{1/2}}\ \exp\left(-\frac{1}{2}(x-\mu)^T\,\Sigma^{-1}(x-\mu)\right)$$

여기서 `|Sigma|` 는 `Sigma` 의 행렬식인데, 여기를 참고하자.

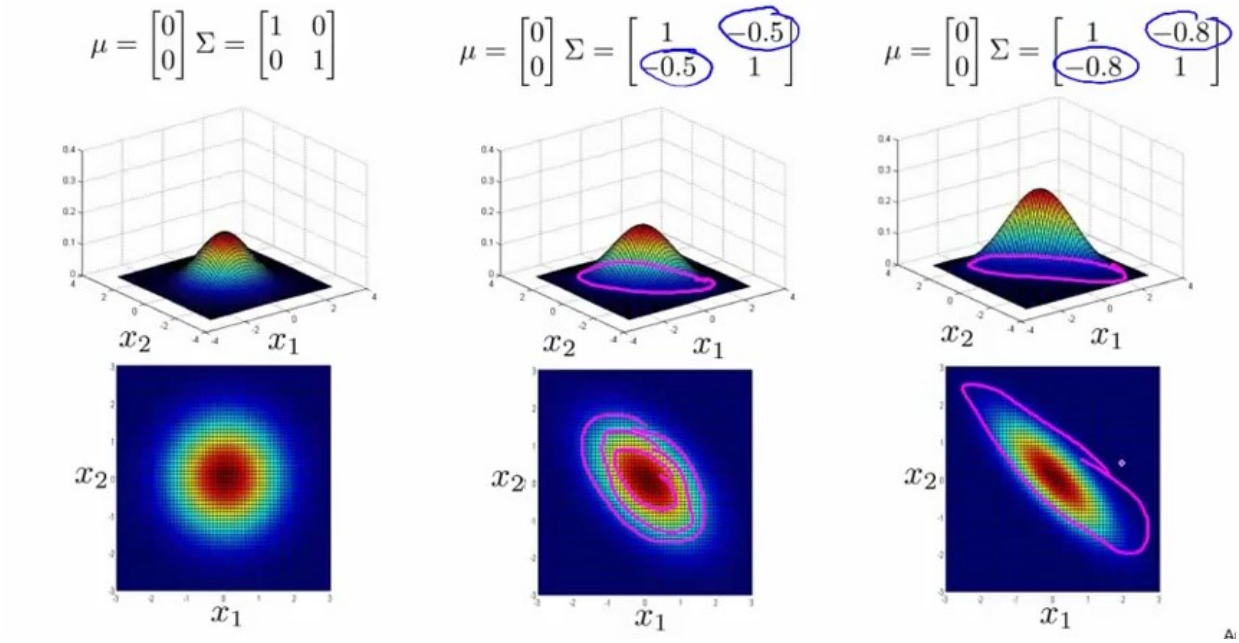
- 행렬
- 행렬식
- 행렬식의 기하학적 의미
- 행렬식과 기하학적 활용

이제 위 식을 이용해서 나온 `p(x)` 를 3차원, 2차원으로 보면





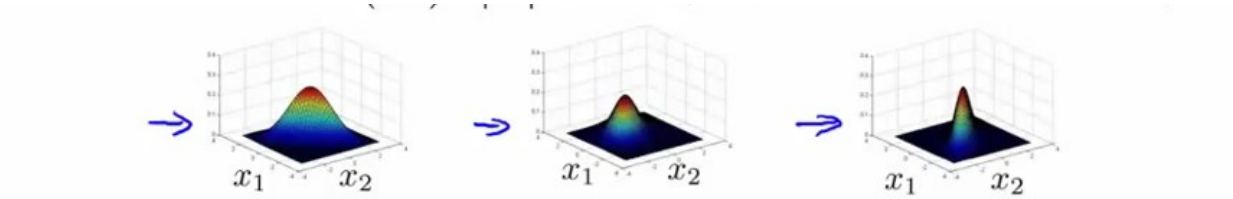
Multivariate Gaussian (Normal) examples



Multivariate Gaussian (Normal) distribution

Parameters μ, Σ $\quad \mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$



Parameter fitting:

Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $x \in \mathbb{R}^n$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

(http://blog.csdn.net/linuxcumt1)

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

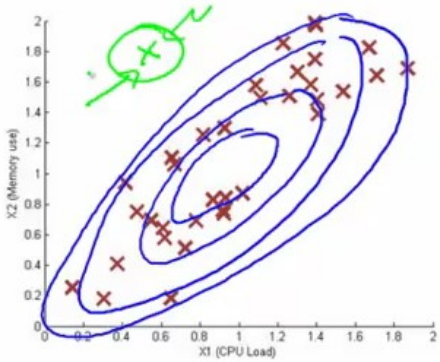
$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

Anomaly detection with the multivariate Gaussian

1. Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

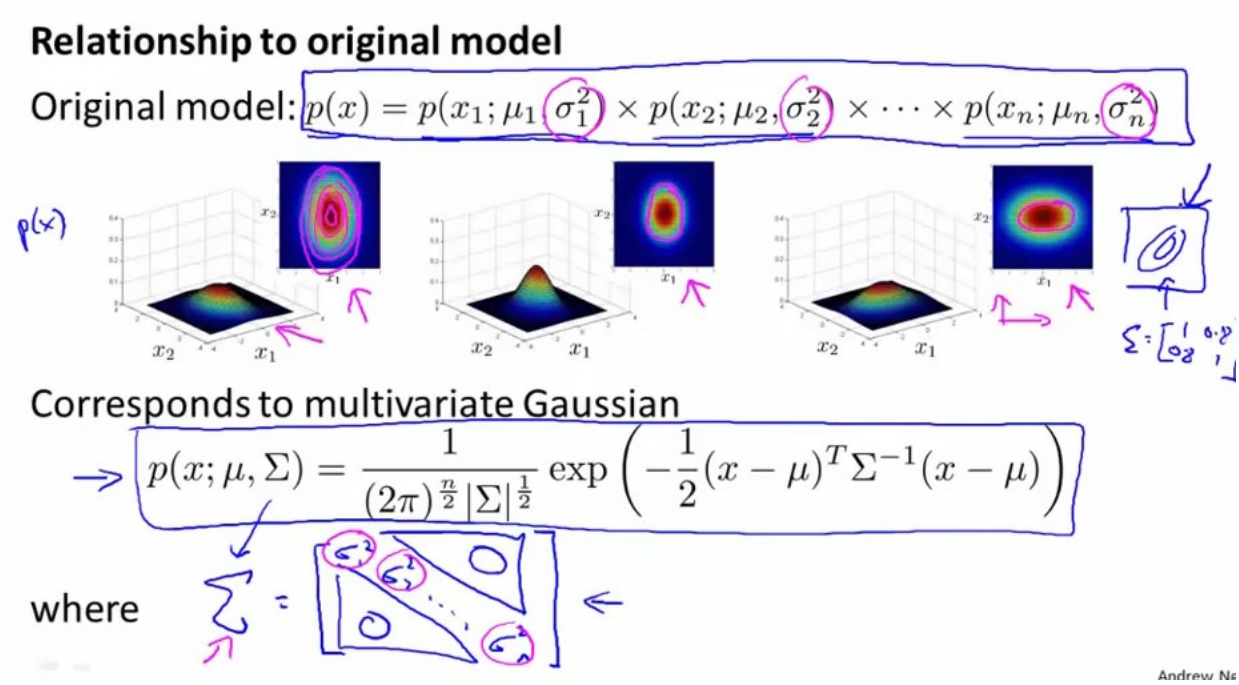


2. Given a new example x, compute

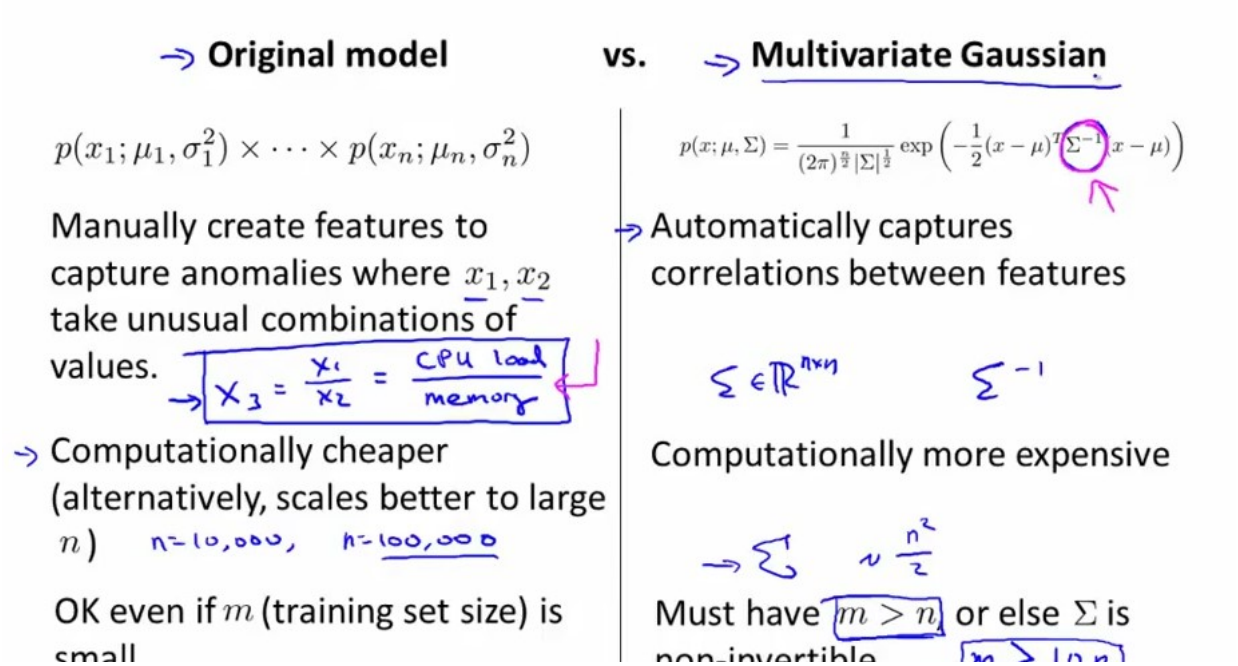
$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Flag an anomaly if $p(x) < \varepsilon$

`u, Sigma` 를 찾아 `p(x)` 를 만들고, 테스트 데이터에 대해 `p(x) < e` 인지 비교한다.



original model 은 *multivariate model* 에서 각 *feature* 간 상관 관계가 없는 (독립), 즉 *covariance matrix* 가 *diagonal matrix* 인 경우다. (



(http://blog.csdn.net/linuxcumt1)

- **Original model**

수동으로 *feature* 를 만들때 사용할 수 있다. 또는 적은 연산을 원할때, 다시 말해서 `n` 이 커서 연산이 무지막지하게 클 때 좋다.

`m` 이 작아도 쓸 수 있다.

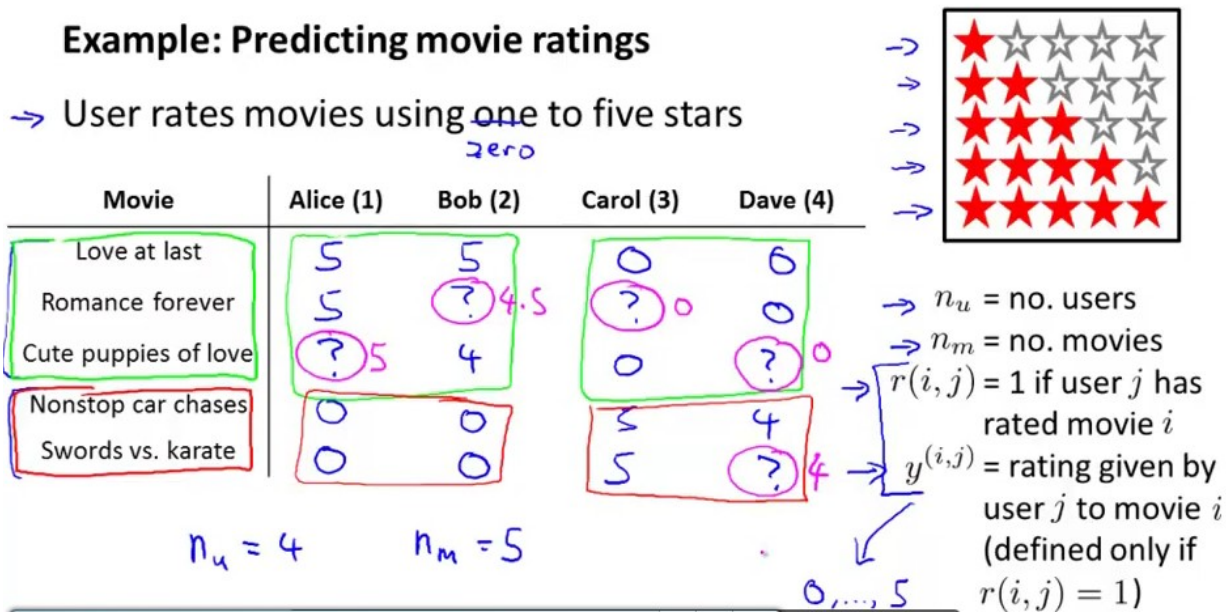
- **Multivariate Gaussian**

계산 비용이 비싸지만, 자동으로 *feature* 간 상관관계를 모델에 포함시킨다.

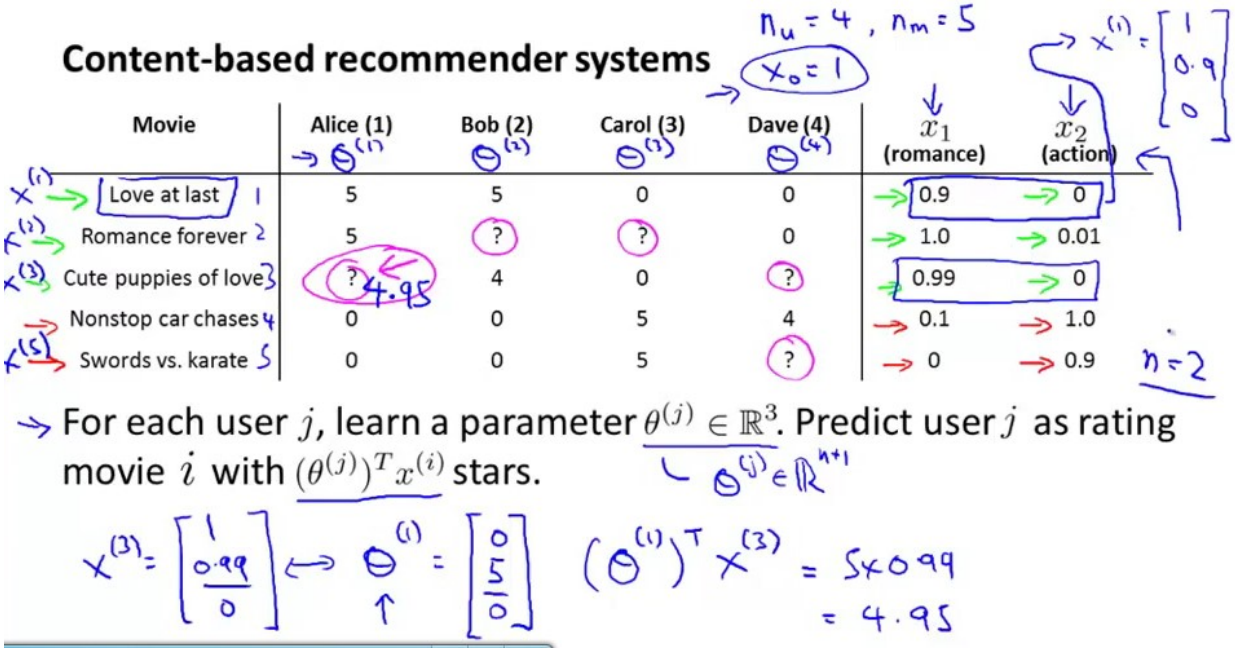
`Sigma` 가 *invertible* 이기 위해서는 `m > n` 이어야 한다. 실제로는 `m` 이 `n` 보다 훨씬 클 때 사용하는 경우가 많다. (e.g. `m >= 10n`)

만약에 `m > n` 인데, `Sigma` 가 *non-invertible* 이면 *redundant feature* 가 있는 경우니 확인해 보자. (흔한 오류라고 함)

Recommender System

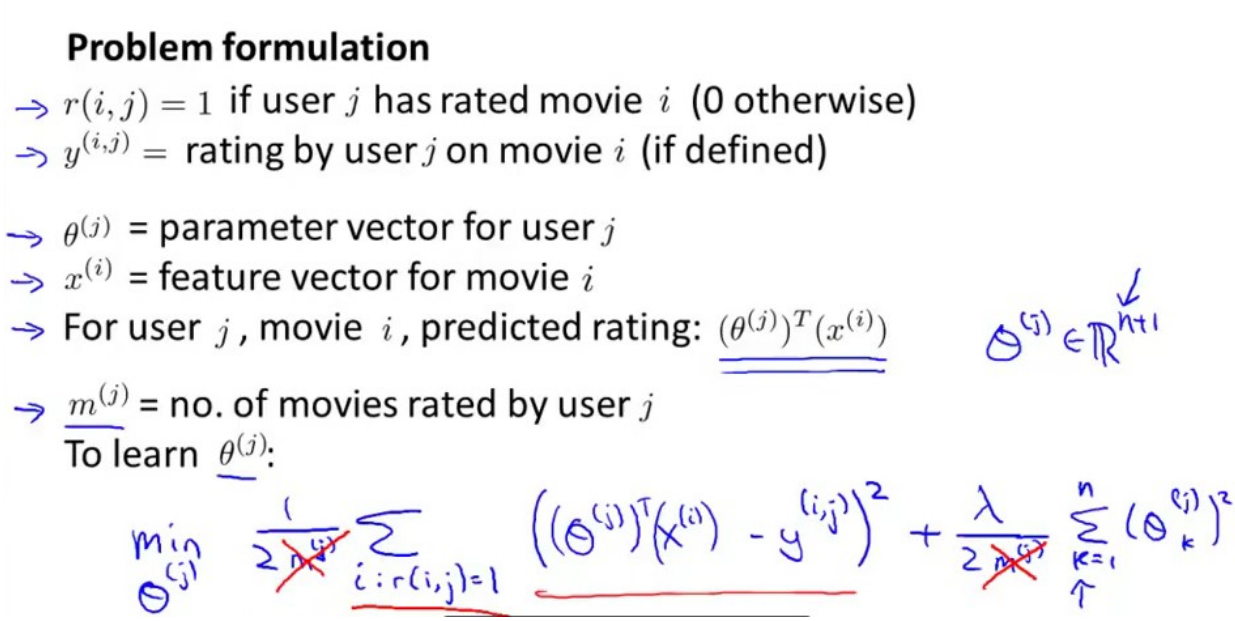


Content Based Recommendations



(http://blog.csdn.net/linuxcumt1)

위 슬라이드는 유저 `j` 로 부터 `theta^(j)` 를 얻어, *feature* `x` 와 곱함으로써 *linear regression* 문제로 변경했다.

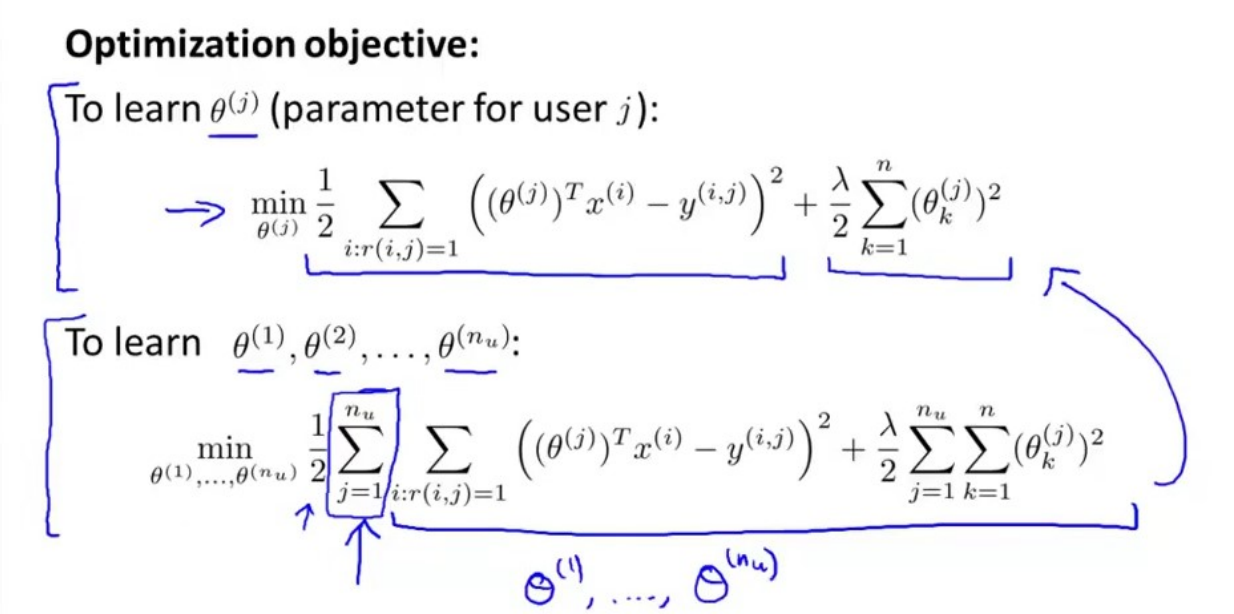


`theta^(j)` 는 어떻게 훈련시킬까?

$$min_{\theta^{(j)}} \sum_{i:\,(ri,j)=1} \frac{1}{2m^{(j)}} [(\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}]^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

여기서 `m^(j)` 는 유저 `j` 에 의해 점수를 받은 영화의 수인데, 어차피 상수이므로 제거하면

$$min_{\theta^{(j)}} \sum_{i:\,(ri,j)=1} \frac{1}{2} [(\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}]^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

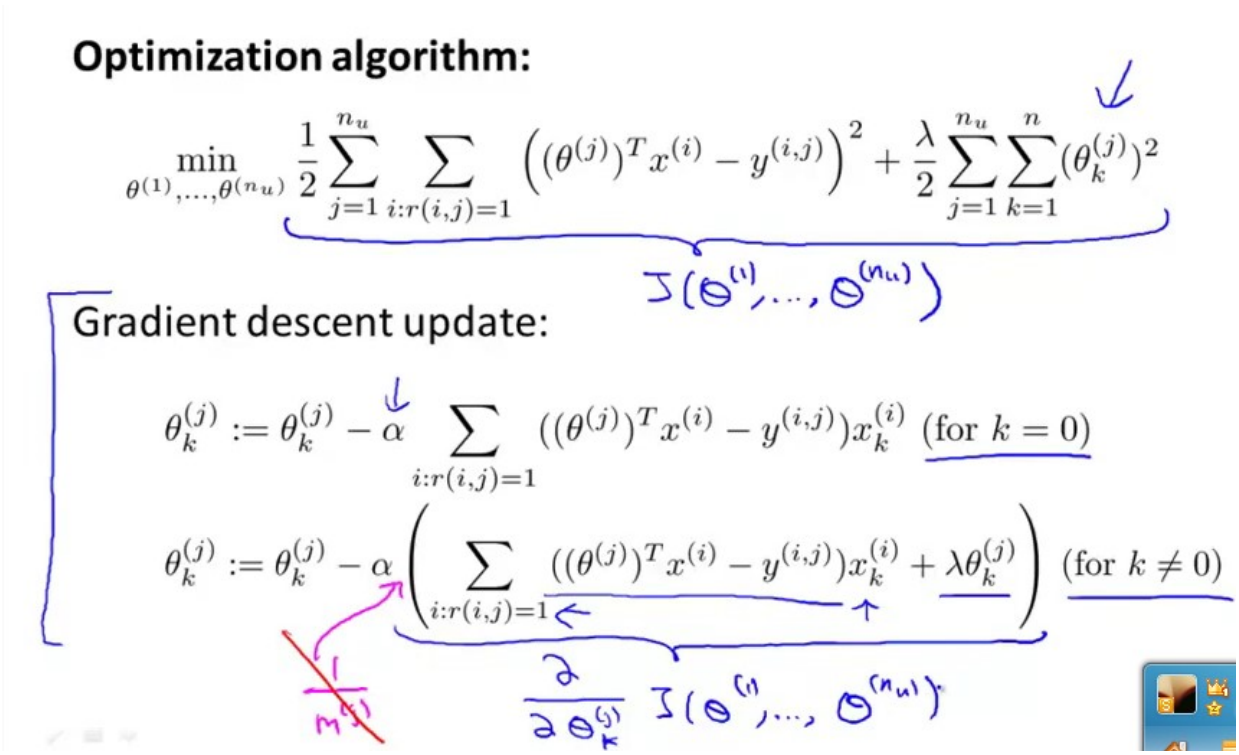


(http://blog.csdn.net/linuxcumt1)

이 때 각 유저마다의 `theta(j)` 를 합 해 최소화 시키는 방식으로 전체 `theta` 를 훈련시킬 수 있다.

$$min_{\theta^{(j)},..\theta^{(n_u)}}$$

$$= \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:\,(ri,j)=1} [(\theta^{(j)})^T(x^{(i)}) - y^{(i,j)}]^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$



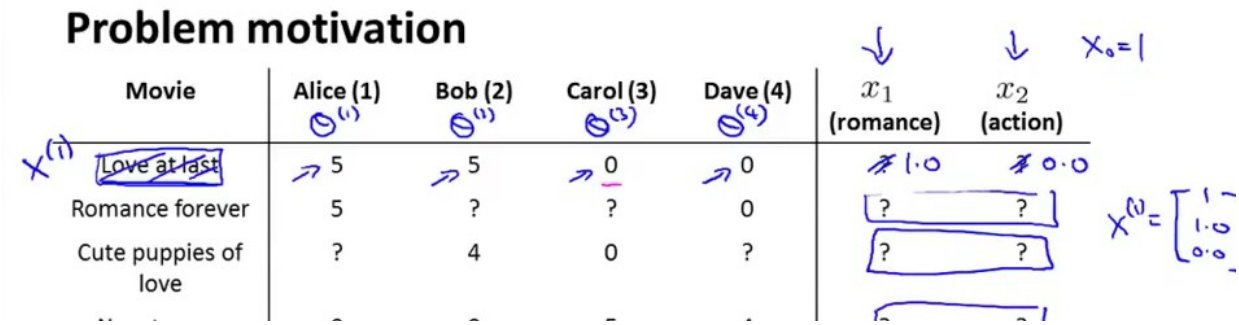
(http://blog.csdn.net/linuxcumt1)

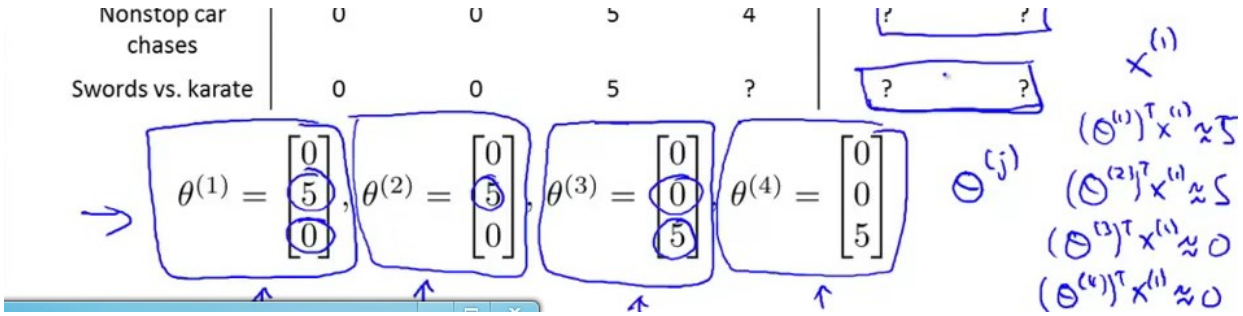
gradient descent 는

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:\ r(i,j)=1}((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})x_k^{(i)} \ (for\ k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:\ r(i,j)=1}((\theta^{(j)})^T(x^{(i)}) - y^{(i,j)})x_k^{(i)} + \lambda\theta_k^{(j)} \ (for\ k \neq 0)$$

Collaborative Filtering





(http://blog.csdn.net/linuxcumt1)

content-based recommendation 에서 *feature* 를 구하긴 사실 어려운 일이다. 누가 이 영화가 얼마만큼 로맨스고, 아닌지를 판별해줄까?

문제를 좀 변경해서, 만약에 유저로부터 `theta(j)` 를 얻어낼 수 있다면 그 정보로부터 *feature* `x(i)` 를 추출할 수 있다. 왜냐하면 `(\theta^(j))^T * x^(i) ~ y^(i, j)` 이기 때문이다.

Optimization algorithm

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

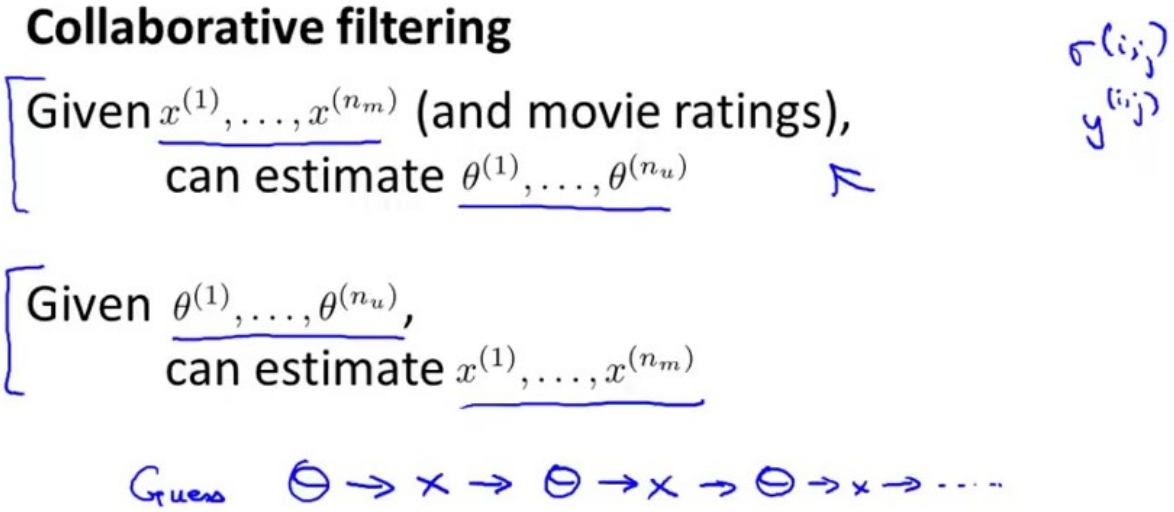
$$\min_{x^{(1)},\ldots,x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

`x^(i)` 를 얻기 위해,

$$min_{x^{(j)},\ldots x^{(n_m)}}$$

$$= \frac{1}{2} \sum^{n_m} \sum [(\theta^{(j)})^T (x^{(i)}) - y^{(i,j)}]^2 + \frac{\lambda}{2} \sum^{n_m} \sum^{n} (x_k^{(i)})^2$$

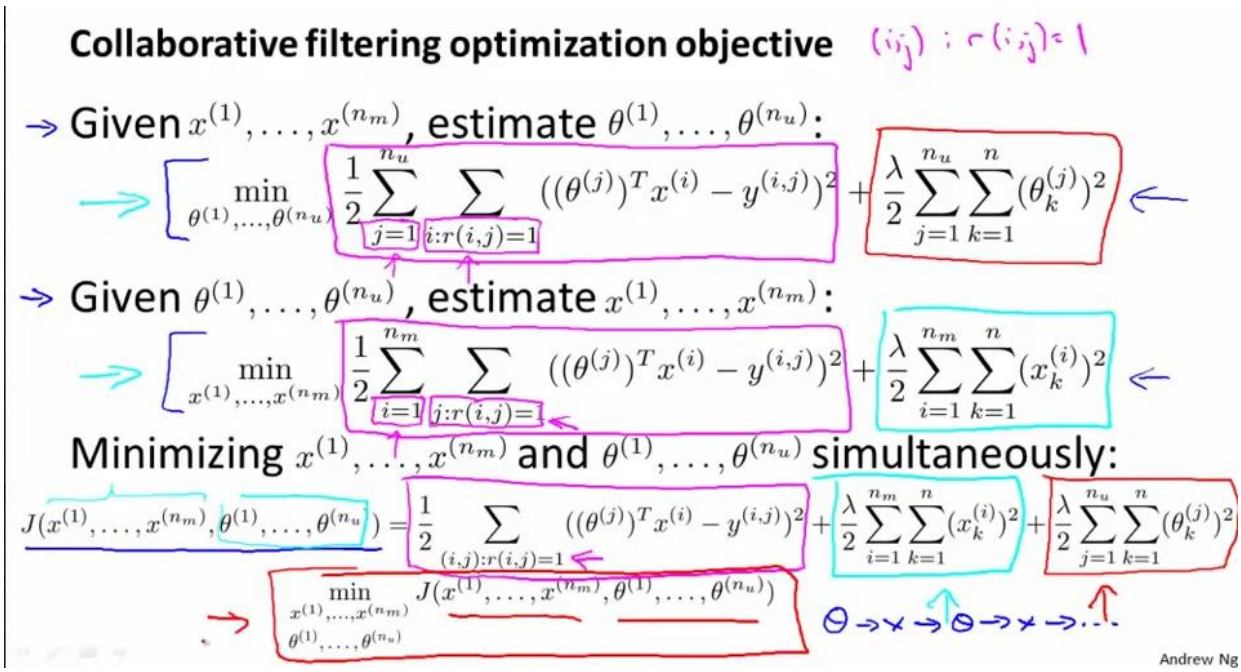
$$2 \sum_{i=1} \sum_{i:\, r(i,j)=1} \qquad 2 \sum_{i=1} \sum_{k=1}$$



(http://blog.csdn.net/linuxcumt1)

- `theta` 가 주어지면 `x` 를 훈련할 수 있고
- `x` 가 주어지면 `theta` 를 훈련할 수 있다.

따라서 최초의 랜덤 `theta` 에 대해 `x` 를 훈련하고, 다시 `theta` 를 훈련하고, 반복하면 된다.

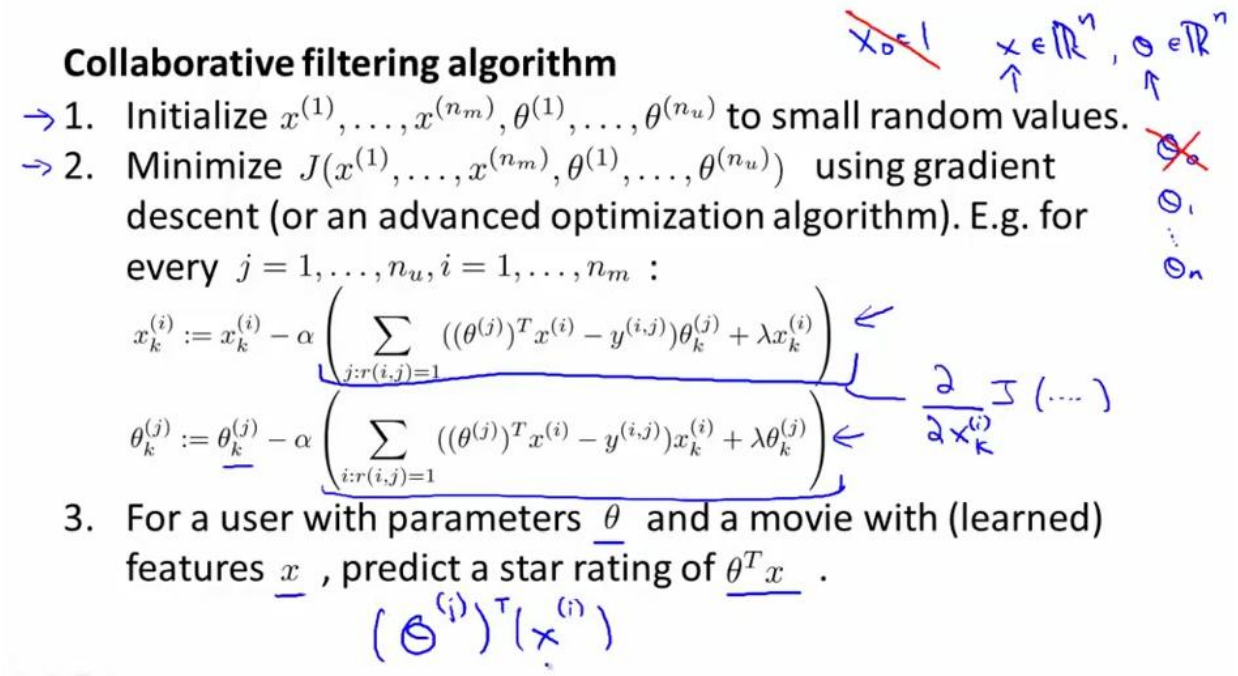


`theta` 와 `x` 를 반복해서 훈련시키는 것보다, 동시에 훈련시키는 것이 좀 더 효율적이다. 따라서

$$J(x^{(j)}, \cdots x^{(n_m)}, \theta^{(i)}, \cdots \theta^{(n_u)})$$

$$= \frac{1}{2} \sum_{(i,j):\ r(i,j)=1} [(\theta^{(j)})^T (x^{(i)}) - y^{(i,j)}]^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

를 최소화 시키면 된다. 참고로 `x_0` 은 *collaborative filtering* 에서 필요가 없다. 알고리즘 자체가 *feature* 를 직접 찾아내니 *hard coded* 된 *feature* 는 사용하지 않는다.



(http://blog.csdn.net/linuxcumt1)

(1) `x`, `theta` 를 작은 값으로 초기화 한다.

이는 *symmetry breaking* 을 하기 위함이다. 작은 랜덤값들로 초기화 하여 `x^(i)` 가 서로 다른 값들을 가지도록 도와준다.

(2) *cost function* `J` 를 *gradient descent* 등으로 최소화 시킨다.

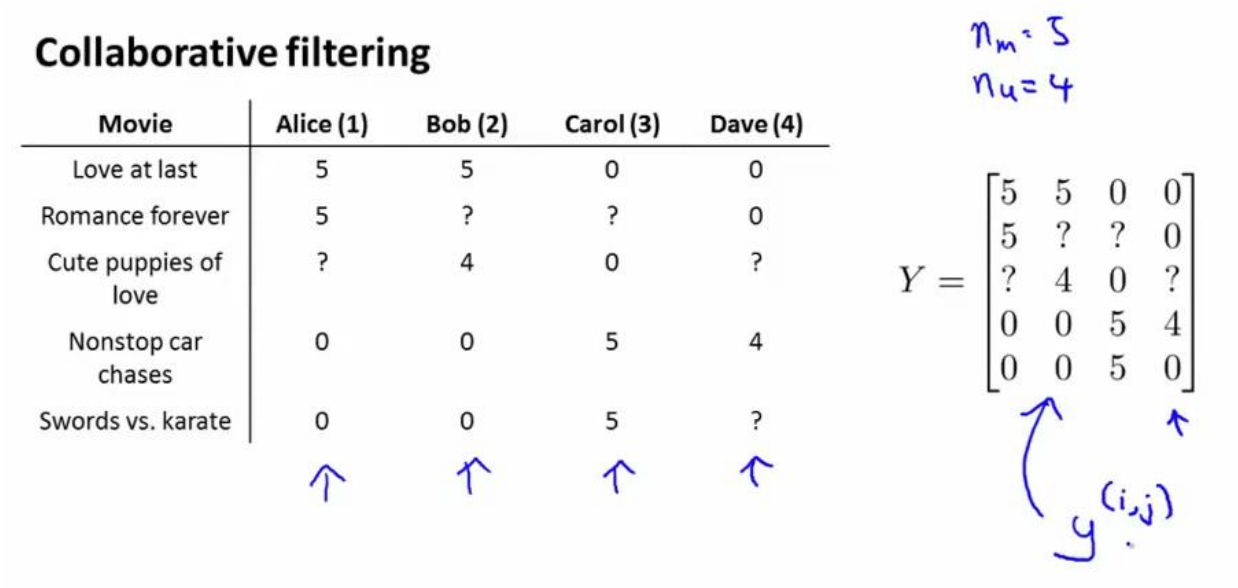
$$x_k^{(i)} := x_k^{(i)} - \alpha \sum [(\theta^{(j)})^T (x^{(i)}) - y^{(i,j)}] \theta_k^{(j)} + \lambda x_k^{(i)}$$

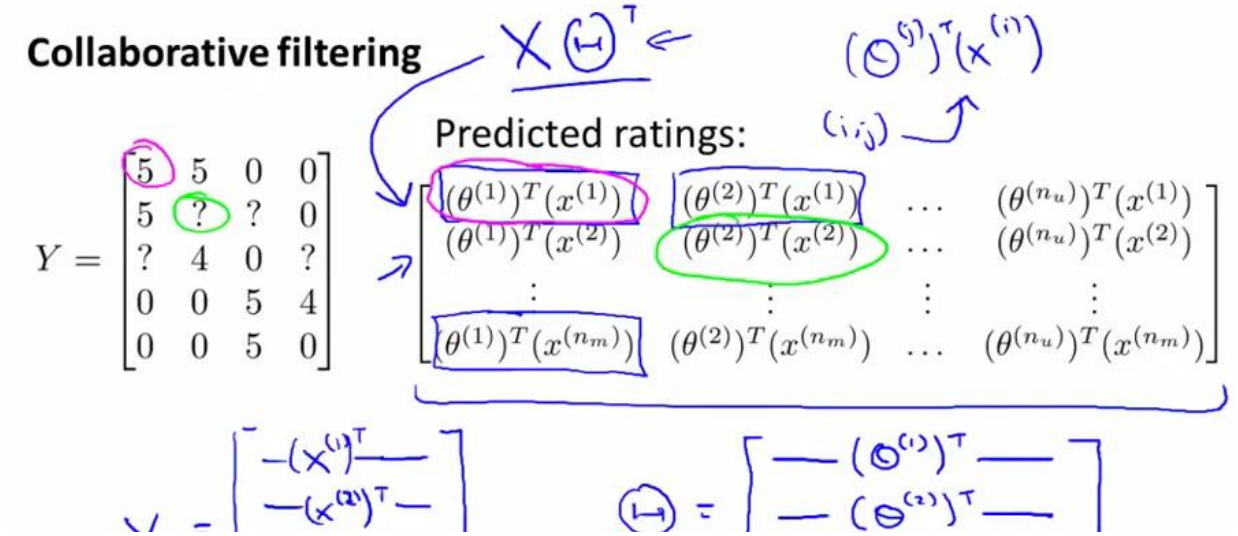
$$\sum_{j:\, r(i,j)=1}$$

$$\theta_k^{(i)} := \theta_k^{(i)} - \alpha \sum_{i:\, r(i,j)=1} [(\theta^{(j)})^T (x^{(i)}) - y^{(i,j)}]_k^{(i)} + \lambda \theta_k^{(j)}$$

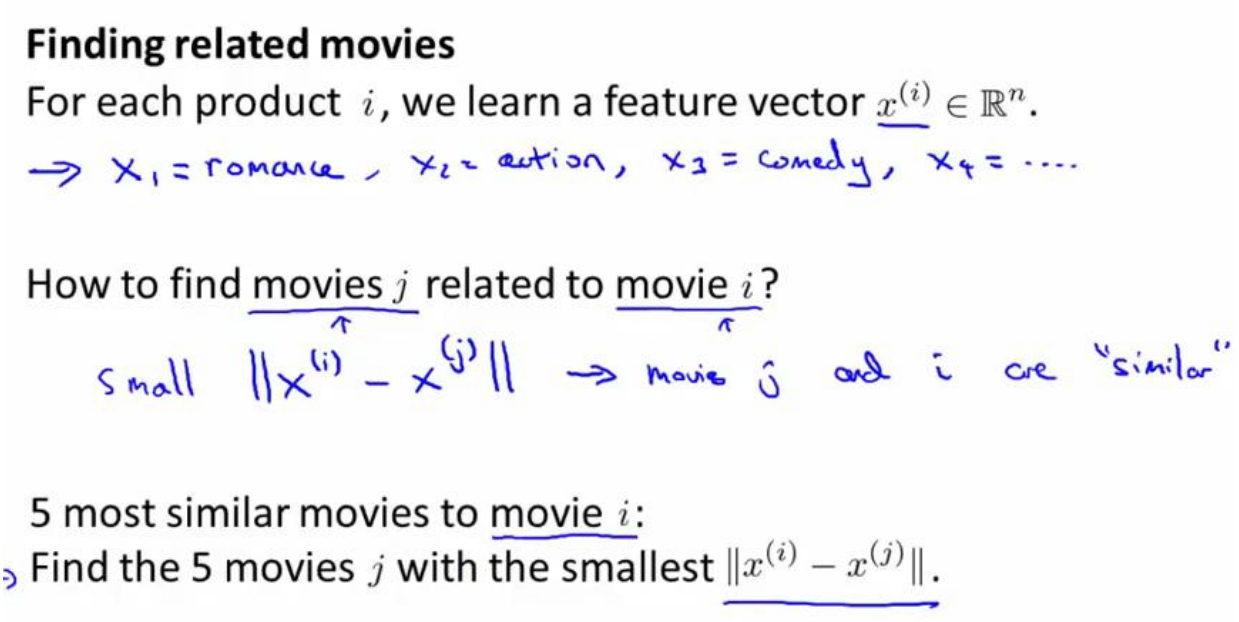
(3) 유저의 *parameter* `theta` 와 영화의 *feature* `x` 에 대해 `theta^T * x` 를 이용해 예측하면 된다.

Vectorization: Low Rank Matrix Factorization





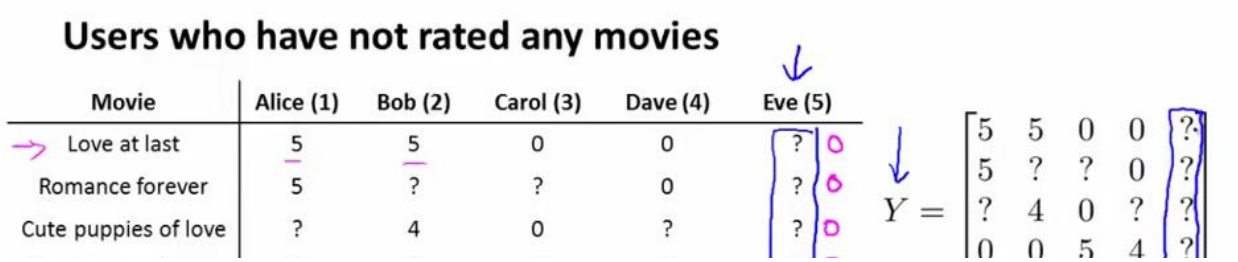
collaborative filtering 은 *low rank matrix factoriazation* 이라 부르기도 한다. 위 슬라이드처럼 `X, THETA` 를 구성하고 `X * THETA^T` 를 구하면 된다.

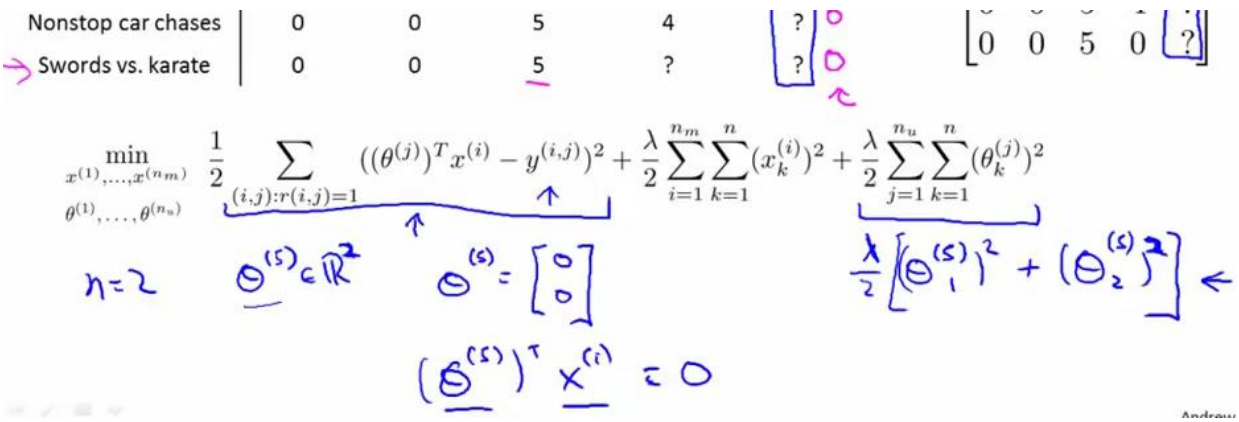


(http://blog.csdn.net/linuxcumt1)

low rank matrix factorization 을 이용해서 *feature* 를 찾으면, 두 영화 `i, j` 가 얼마나 유사한지 `||x^(i) - x^(j)||` 으로 판단할 수 있다.

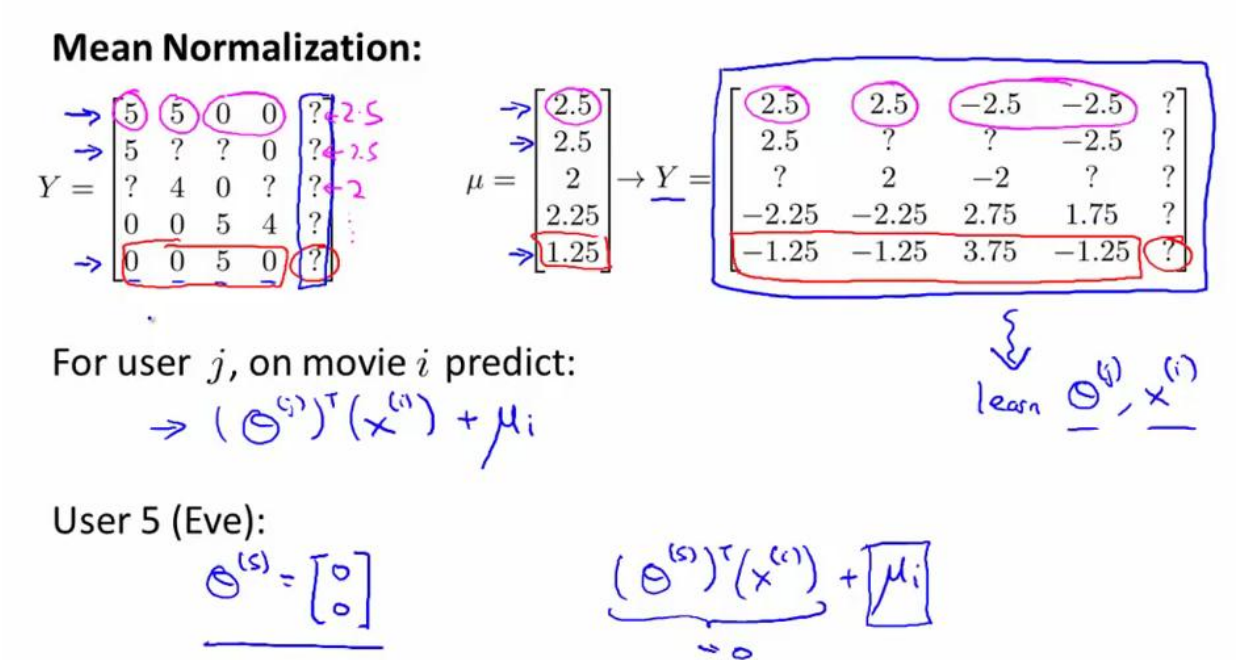
Implementation Detail: Mean Normalization





만약 위 슬라이드의 `Eve` 처럼 아무 영화도 평가 안한 사람에게는, `theta` 가 `0` 으로 나온다. (첫번째 *term* 이 `0` 이고, *regularization term* 은 `theta` 를 최소화한다.)

그렇게 되면, 어떤 영화도 높은 *rating* 을 받을 수 없으므로 (`theta^T * x`). 추천할 거리가 없다. 이건 별로 좋은 상황이 아닌데, *mean normalization* 을 이용하면 이 문제를 해결할 수 있다.



(http://blog.csdn.net/linuxcumt1)

mean normalized 데이터를 이용하면, 추천 안한 사람이 `theta = 0` 을 갖더라도, 남들이 추천한 선호도 `u` 에 따라서 영화를 추천받을 수 있다.

잘보면 *feature scaling* 과는 다르게 특정 *range* 로 나누질 않는데, 이건 이미 *rating* 자체가 일정 범위 `1-5` 를 갖기 때문이다.

References

(1) *Machine Learning* by **Andrew NG**
(2) http://blog.csdn.net/linuxcumt
(3) http://blog.csdn.net/abcjennifer
(4) http://ghebook.blogspot.com
(5) http://darkpgmr.tistory.com